RESEARCH-ARTICLE

Enhanced Web Application Security Through Proactive Dead Drop Resolver Remediation

JONATHAN FULLER

MINGXUAN YAO, Georgia Institute of Technology, Atlanta, GA, United States

SAUMYA AGARWAL, Georgia Institute of Technology, Atlanta, GA, United States

SRIMANTA BARUA, Georgia Institute of Technology, Atlanta, GA, United States

TALEB HIRANI, Georgia Institute of Technology, Atlanta, GA, United States

AMIT KUMAR SIKDER, Iowa State University, Ames, IA, United States

View all

Open Access Support provided by:

Georgia Institute of Technology

Iowa State University

PDF Download
3719027.3744860.pdf
26 January 2026
Total Citations: 0
Total Downloads: 1304

Published: 19 November 2025

Citation in BibTeX format

CCS '25: ACM SIGSAC Conference on Computer and Communications Security
October 13 - 17, 2025
Taipei, Taiwan

Conference Sponsors:
SIGSAC

Enhanced Web Application Security Through Proactive Dead Drop Resolver Remediation

Jonathan Fuller*
United States Military Academy
West Point, New York, USA

Mingxuan Yao*
Georgia Institute of Technology
Atlanta, Georgia, USA

Saumya Agarwal
Georgia Institute of Technology
Atlanta, Georgia, USA

Srimanta Barua
Georgia Institute of Technology
Atlanta, Georgia, USA

Taleb Hirani
Georgia Institute of Technology
Atlanta, Georgia, USA

Amit Kumar Sikder
Iowa State University
Ames, Iowa, USA

Brendan Saltaformaggio
Georgia Institute of Technology
Atlanta, Georgia, USA

Abstract

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

CCS Concepts

• **Security and privacy** → **Malware and its mitigation**.

Keywords

Malware; Internet Dead Drops; Botnet Counteraction

ACM Reference Format:
Jonathan Fuller*, Mingxuan Yao*, Saumya Agarwal, Srimanta Barua, Taleb Hirani, Amit Kumar Sikder, and Brendan Saltaformaggio. 2025. Enhanced Web Application Security Through Proactive Dead Drop Resolver Remediation. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25), October 13–17, 2025, Taipei, Taiwan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3719027.3744860

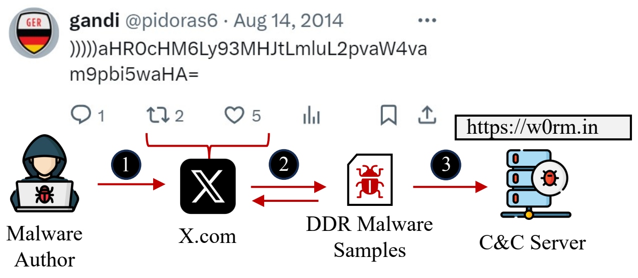

gandi @pidoras6 · Aug 14, 2014
)))))aHR0cHM6Ly93MHJtLmluL2pvaW4va
m9pbi5waHA=

Figure 1: Razy's [6] DDR workflow. ❶: The malware author *manipulates* the C&C server address and *posts* it on X.com as a dead drop. ❷: Razy executes on the victim systems and *fetches* the dead drop. ❸: Razy *resolves* the dead drop content to connect to the C&C server.

1 Introduction

Dead Drop Resolver (DDR) is a technique developed by malware authors to evade traditional takedown methods for Command and Control (C&C) servers [47, 4, 73, 85, 105, 111, 81, 42, 100, 11, 115]. For example, Figure 1 shows the workflow of the *razy* [6] malware. *Razy manipulates* (i.e., encodes or encrypts) its C&C server address and hosts it on X.com, serving as "dead drop" (❶). Once executed on the victim machine, *razy* fetches ❷ and decodes ❸ the dead drop content to resolve its C&C server address, confirming it as a DDR malware sample.

Current remediation strategies fail for DDR malware samples because web app providers respond to individual dead drops that a DDR malware sample is actively using ❸. However, malware authors can reuse manipulation techniques to encode new C&C addresses, creating polymorphic dead drops. As a result, the reactive approach misses opportunities to mitigate dead drops early in the

*Authors contributed equally.

DDR lifecycle ❶. This research proposes a proactive approach, where web app providers can scan their platforms to identify and neutralize dead drops.

Malware authors are increasingly exploiting web apps to host dead drops [112], yet state-of-the-art techniques lack the scalability needed for proactive remediation by web app providers. The current methods primarily rely on manual inspection to understand how malware authors encode C&C addresses hosted on web apps [91, 93, 98] or stored as blockchain transactions [89, 41]. Proposals such as submitting new transactions to update the most recent blockchain data to sinkhole C&C addresses [97] demand extensive manual effort and remain ad-hoc. Similarly, web app providers can try to detect dead drops by profiling incoming connections from DDR malware samples to their platforms. However, existing malicious connection detection techniques [82, 47, 66, 42, 64, 43, 39] are ineffective in the DDR context. This is because web apps enforce the use of specific protocols for all connections, making it impossible to distinguish between legitimate and malicious connections.

Adding to these difficulties, web app providers face technical challenges in combating DDR malware. One key issue is the wide variety of manipulation techniques (e.g., BASE64), making it impractical to decode all content using every method. Even worse, C&C addresses are often encoded using multiple layers of techniques, some requiring specific parameters (e.g., XOR). In fact, single dead drops can employ three or more manipulation layers (§5.3.1), rendering brute forcing infeasible. However, our analysis of DDR incidents shows DDR malware samples must be equipped with the precise combination and configuration of de-manipulation techniques. This led to our first key insight: Understanding how DDR malware samples decode dead drop content allows web app providers to derive reusable "recipes" to proactively identify similarly encoded C&C addresses.

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

Even with the DDR capability confirmed, identifying the exact de-manipulation techniques and their order requires manual analysis [97, 89, 45]. Nevertheless, our research discovered a third key insight: De-manipulation routines applied to fetched content can reduce to mathematical operations. By abstracting these formulas and comparing them to known patterns, web app providers can confirm the de-manipulation techniques used and generate effective recipes for proactive remediation.

To explore our key insights, we studied DDR malware samples with support of Netskope, the leading Secure Access Service Edge (SASE) provider, which delivers cloud security and networking to more than 30% of Fortune 100. We developed VADER for proacti**V**e de**A**d **D**rop r**E**solver **R**emediation. VADER is a web app and malware analysis framework to automatically detect and proactively uncover dead drops. VADER systematically explores a DDR malware sample to *localize the DDR logic* (§3.1). It then identifies de-manipulation algorithms by analyzing the symbolic expressions generated during its execution. However, a significant challenge lies in accurately isolating the relevant segments of these expressions for precise matching. To address this, VADER employs an innovative *Input/Output (IO) boundary isolation* technique (§3.2), that delineates the boundaries of routines within the symbolic expressions. This approach enables VADER to effectively isolate and match expressions, deriving de-manipulation recipes (§3.3). Through this recipe-based pattern-discovery approach, VADER can efficiently uncover C&C addresses concealed within dead drops.

We deployed VADER on 100k malware samples from 2012 to 2022, identifying 8,906 DDR malware samples and 273 dead drops. Pastebin was the most prevalent platform, accounting for 68% of cases. This underscores the importance of VADER-generated recipes for proactive dead drop discovery. As of this submission, we have removed 94.4% of identified dead drops through collaboration with web app providers, disrupting at least 6,674 DDR malware samples by eliminating the dead drops they relied on. Applying these recipes to network traffic datasets from open-source projects [83], we uncovered four additional abused web apps and 72 previously undisclosed dead drops. We also presented two case studies on de-manipulation recipe complexity and a real-world forensic analysis of a prominent remote access trojan. The VADER framework can be found at: https://github.com/CyFI-Lab-Public/VADER.

2 Overview

2.1 Running Example - The Mudrop Malware

In VADER 's practical demonstration to combat DDR malware and uncover more dead drops, we use *mudrop* as the running example. Imagine this forensic scenario: the WordPress [107] security team receives an alert from FIRST [33], a global cybersecurity organization, about a new malware sample, *mudrop*. Flagged as a potential threat to WordPress, *mudrop* is shared by FIRST as part of its collaborative security effort.

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

At this point, WordPress must answer several forensic questions:

Q1: Is *mudrop* a DDR malware sample?
Q2: Does *mudrop* use WordPress as a dead drop?
Q3: How does *mudrop* decode its dead drop?
Q4: Are there other dead drops hosted on WordPress?

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

```
f237769666e6f636f2336313e28393e2039313e2838313f2f2a307474786
```

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

Table 1: VADER's Mudrop Proactive Discovery Results.

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

To address these research challenges and answer the aforementioned questions, VADER employs novel techniques to: (Q1 & Q2) *localize* DDR logic within the malware (§3.1); (Q3) apply de-manipulation to dead drop content (§3.2); and formulate a de-manipulation *recipe* (§3.3). (Q4) Then, the recipe can be applied to web app content or network traffic, proactively identifying *previously undisclosed* dead drops. This multiplies the effectiveness of DDR remediation—a contribution not yet explored in prior research. Moreover, WordPress can share *mudrop*'s recipe with FIRST and web app providers to extend remediation.

Once the *mudrop* sample is provided, VADER identifies its DDR logic, notifying the WordPress provider that the *mudrop* sample retrieves dead drop content from a WordPress post and decodes it to obtain the C&C address, as shown in Table 1. The provider can then remove the dead drop content. However, providers still lack insight into how *mudrop* decodes the content, which is key to proactively finding hidden dead drops. By analyzing the *mudrop's* DDR logic, VADER tracks the decoding process, revealing layered Base16 encoding and character rotation algorithms (i.e., the recipe) to extract the dead drop content (see Table 1, *VADER's Mudrop Analysis Output*).

2.2 Proactive Dead Drop Discovery

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

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

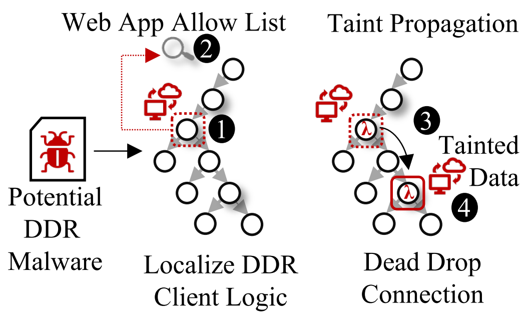


Figure 2: DDR Logic Localization.

2.3 VADER in Practice

Web App Providers + VADER. Section §2.2 shows how VADER's recipes can enhance remediation by proactively uncovering dead drops. While our lab setting limits access to web app platforms, providers are better positioned to scan their own platforms at scale. Platforms like Discord already scan hosted files proactively [25], and Google uses its Content Safety API to detect child sexual abuse material at scale [101]. Additionally, Roblox, Discord, and Reddit also work with law enforcement to identify extremist users via advanced scanning [46]. We are therefore confident that these providers can use VADER to proactively detect dead drops.

Intrusion Detection System + VADER. Intrusion Detection Systems (IDS) can integrate VADER. When malware is detected, VADER verifies its DDR logic and extracts recipes, which IDSs can use to analyze network traffic and de-manipulate embedded payloads. If any IPs or URLs are found, analysts can use internal tools to confirm maliciousness [77], enabling proactive dead drop discovery and strengthening network protection. This is shown in §5.2, where VADER's recipes were applied to network traffic. By using these recipes to de-manipulate payloads, we proactively uncovered 72 dead drops across 11 web apps, hiding 67 C&C addresses. Our collaboration with web app providers led to the removal of 94.4% of these proactively identified dead drops.

3 Methodology

VADER uses a three-phase approach: *DDR logic localization* (§3.1) takes a potential DDR malware sample as input to confirm DDR capabilities. The output is a symbolic expression generated during DDR malware sample exploration. VADER then *isolates de-manipulation boundaries* (§3.2) to segment the symbolic expression for comparison with a set of reference de-manipulation algorithms for *recipe* identification (§3.3). Finally, a *recipe-based pattern-discovery* solution demonstrates how the recipe can be used to proactively uncover previously undisclosed dead drops.

3.1 DDR Logic Localization

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

DDR Client Connection. During VADER's analysis, VADER intercepts the DDR malware sample's invocation of network APIs used for web app connection (Figure 2, ❶). Web app connection is determined by comparing the connection target (e.g., *mudrop* uses `wordpress.com`) against a predefined list of allowed web app candidates (Figure 2, ❷). This list is based on Tranco [62][1], allowing us to identify only benign websites that support DDR[2]. After the web app connection is established, the DDR malware sample retrieves the encoded content from the dead drop (e.g., `blizzartone` for *mudrop*).

Dead Drop Connection. To confirm the dead drop connection, VADER injects symbolic data into the memory location allocated by the malware to store the fetched content (e.g., *mudrop* uses `lpBuffer` of `WinHttpReadData`) and tags it (Figure 2, ❸). VADER uses concolic taint propagation to track the tag. However, de-manipulation algorithms generally contain loops that result in multiple states for exploration, thereby increasing computational complexity when using concolic execution. De-manipulating the encoded content requires that each character be transformed into cleartext. Thus, the content length (e.g., `lpdwNumberOfBytesRead`) determines how many iterations of character transformation are necessary. To ensure tractable analyses amid multi-state exploration, VADER uses state prioritization. Specifically, VADER prioritizes unexplored code and selects states corresponding to new code regions. For example, if VADER explores a de-manipulation loop, it will generate two states:

(1) $i <$ sizeof(`lpdwNumberOfBytesRead`)

(2) $i >$ sizeof(`lpdwNumberOfBytesRead`)

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

(1_1) $i + 1 <$ sizeof(`lpdwNumberOfBytesRead`)

(1_2) $i >$ sizeof(`lpdwNumberOfBytesRead`)

Since state (1) and (1_2) explore the same code in two different states, state (1_2) is de-prioritized. This technique ensures that VADER can efficiently handle loops while ensuring that de-manipulation algorithms are fully explored as it prioritizes state (1) until complete.

During exploration, the tag emerges in a network connection API (e.g., *mudrop* uses `pswzServerName` for `WinHttpConnect`) Figure 2, ❹. Importantly, what remains is a symbolic expression with all data computations performed on the manipulated content.

3.2 De-Manipulation IO Boundary Isolation

This phase aims to isolate de-manipulation algorithms in the symbolic expression using their input and output data to mark their boundary. De-manipulation includes decryption (e.g., AES) and decoding (e.g., Base16). Our preliminary investigation suggests 14 de-manipulation (nine decoding, five decryption) algorithms common in malware (Table 2).

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

[1]Available at https://tranco-list.eu/list/7X9VX.

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

Table 2: Common Data Manipulation Algorithms.

Decryption & Decoding Algos	References
Exclusive OR (XOR)	[10, 31, 35, 78, 99]
AES	[60, 76]
DES	[20, 44]
Data Protection API	[44]
RC4	[10]
String to Int, Int to String	[90]
Character Rotation	[78, 35]
Character Subtraction	[78]
Base 16	[29, 91, 78]
Base 32	[32]
Base 64	[6, 35, 78, 29, 93]
Base 85	[103, 19]
String Parsing	[6, 90, 35, 75, 91]

Symbolic expressions, especially when subjected to malware-induced mathematical manipulations, lack clear observable boundaries, complicating the partitioning of multiple de-manipulation algorithms and potentially hindering dead drop discovery. To address this, VADER employs the symbolic expression from DDR logic localization (§3.1) and uses concrete decoder analysis (§3.2.1) and decryption source-to-sink mapping (§3.2.2) to isolate decoders and decryption processes.

3.2.1 Concrete Decoder Analysis.

To isolate decoder boundaries, VADER uses Algorithm 1, which relies on concrete values computed during DDR malware sample execution (concrete values are based on constraints accumulated during execution) to find the input and output (IO) of the decoder. As VADER analyzes the DDR malware sample, Algorithm 1 evaluates memory accesses and concretizes its symbolic contents (Line 4 to Line 6). Next, it iterates through decoders from Table 2 using the concrete data as input (Line 7 to Line 8). Each output is stored in a container for algorithm-to-instruction mapping L for later referencing (Line 9). Thus far, the concretized data C (Line 6), the decoder D (Line 7), the decoded result d (Line 8), and L (Line 9) have been initialized. This process continues throughout VADER's analysis. Concurrently, the algorithm iterates through all previously stored results in L (Line 10 to Line 13) to extract comparative data for boundary isolation. Specifically, if the algorithm finds a previous decoded value (e.g., the C&C address) that matches the current concrete value, then the decoder boundary begins at the instruction of the decoded value to the instruction of the current matching concrete value.

We illustrate Algorithm 1 using *mudrop* in Figure 3. At instruction #4 (*Inst*4), *mudrop* accesses memory and VADER concretizes the memory to *mudrop's* fetched content (Figure 3, ❶). The input is submitted to reference decoder algorithms, and the output (Figure 3, ❷) is stored for later comparison. As execution continues, VADER compares concretized results with previously decoded results to identify a match. If a match is identified

Algorithm 1: De-Manipulation Boundary Isolation

```
   Input: $Malware$, $Decoders \backslash Decryptors = \{D^1 \dots D^{16}\}$
   Output: $DB$ : Container to store De-Manipulation Boundaries
   // Store Algorithm-to-Instruction mapping
1  $L \leftarrow \varnothing$
2  Function DeManipulationBoundary($Malware$) do
3      while $Instruction \leftarrow ExploreMalware(Malware)$ do
4          $I \leftarrow Instruction$
5          if $M \leftarrow MemoryAccessed(I)$ then
6              $C \leftarrow ConcretizeMemoryContent(M)$
               // Iterate 9 Decoder Algorithms
7              foreach $D \in Decoders$ do
8                  $d \leftarrow DecodeConcreteData(D, C)$
9                  $UpdateDecoderMapping(L, I, C, d)$
                   // Iterate previously mapped results
10                 foreach $l \leftarrow L$ 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                         $UpdateBoundary(DB, D, I_p, I)$
14                     end
15                 end
16             end
17         else if $C_i \leftarrow ImportedFuncCall(I)$ then
               // Iterate 7 Decryption Algorithms
18             foreach $D \in Decryptors$ do
                   // Locate Start and End boundaries
19                 if $S, I_s \leftarrow DecryptIO(D, C_i)$ then
20                     $IBoundary \cup (S, I_s)$
21                 else if $E, I_e \leftarrow DecryptIO(D, C_i)$ then
22                     $OBoundary \cup (E, I_e)$
23                 end
24             end
25         end
26     end
       // Create Decryption IO Pairs
27     $DecryptIOPairs \leftarrow MatchIO(IBoundary, OBoundary)$
28     for $Pr \in DecryptIOPairs$ do
29         $D, I_s, I_e \leftarrow Pr$
30         $UpdateDeManBoundary(DB, D, I_s, I_e)$
31     end
32     if $DB$ then
33         $ClosestBoundary(DB)$
34     end
35 end
```

(Figure 3, ❸), the decoder boundary (e.g., Base64) is from *Inst*4 to *Inst*35 (Figure 3, ❹). However, VADER may identify multiple boundaries for each decoder. Thus, the boundary for Base64 is also *Inst*12 to *Inst*35 (Figure 3, ❺). This occurs if a wrapper function is used that does not modify input or output values. Before completing boundary analysis, VADER computes the shortest distance between boundary instructions for boundary isolation (Line 33 in Algorithm 1).

Using the boundary, VADER segments the symbolic expression from §3.1 for verification. An astute reader may consider symbolic expression matching redundant since VADER has already concretely delineated the DDR malware sample's decoders. This is necessary to discriminate two algorithms producing the same output, e.g., if *mudrop's* encoded content `f237769666e6f636f2336313e28393e2039313e2838313f2f2a307474786`

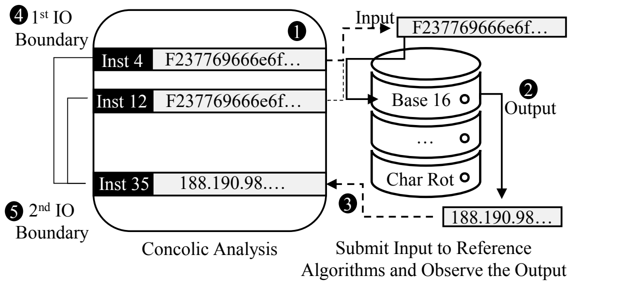


Figure 3: *Mudrop*'s Decoder IO Boundary Isolation.

is decoded with Base16 and character rotation, the result is `http://188.190.98.163/configs` (see Table 1). If we use the same input and XOR with `f25f02e216dcd919c30b5b3dd3baa3cc3aab3dd3b503c919dcd612e20f` we arrive at identical outputs. If the DDR botnet orchestrators post additional encoded messages, relying on only the concretely derived decoders could lead authorities to apply the incorrect de-manipulation recipe. Thus, VADER compares each segmented boundary with its corresponding reference implementation symbolic expression for robust verification.

3.2.2 Decryption Source-To-Sink Mapping. VADER intercepts invoked API for DDR logic localization (§3.1). These interceptions also prove useful to link decryption APIs-in-sequence for analysis. Specifically, while VADER explores the DDR malware sample (Line 3 in Algorithm 1), it evaluates call instructions and selects those that are specific to imported functions (Line 17 in Algorithm 1). If the imported function is an API responsible for instantiating decryption (e.g., `CryptAcquireContext`), VADER updates the *IBoundary* set representing the boundary source (Line 19 to Line 20 in Algorithm 1). If the boundary sink is identified (e.g., `CryptReleaseContext`), VADER updates *OBoundary* (Line 21 to Line 22 in Algorithm 1). To pair sources and sinks, VADER uses *MatchIO*() (Line 27 in Algorithm 1). For example, during concolic exploration, if two states are explored, and the first state invokes a source and sink API, those APIs are paired since they were invoked and intercepted by VADER from the same state. This ensures that APIs invoked in different paths are not incorrectly paired. Finally, these pairs are used to update our container of all de-manipulation boundaries for analysis (Line 28 to Line 28 in Algorithm 1).

3.3 De-Manipulation Recipe Identification

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

Segmenting Expressions. Algorithm 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

Algorithm 2: Symbolic Expression Matching

```
   Input: $DB$ : Container to store Decoder Boundaries
   Output: $DR$ : Decoding Recipe
1  $M_\lambda$ ← Malware Symbolic Expression
2  $RI$ ← Symbolic Expressions for Reference Implementations
3  Function SymbolicExpressionMatching(DB)
4      foreach $b \in DB$ do
           // Extract the decoder and start & end boundary
              addresses
5          $D, I_s, I_e \leftarrow b$
6          $s \leftarrow SegmentExpression(M_\lambda, I_s, I_e)$
           // Assign decoder symbolic expression
7          $ri \leftarrow RI.index(D)$
8          if $dm \leftarrow CompareExpressions(s, ri)$ then
9              $DR.add(dm)$
10         end
11     end
12 end
```

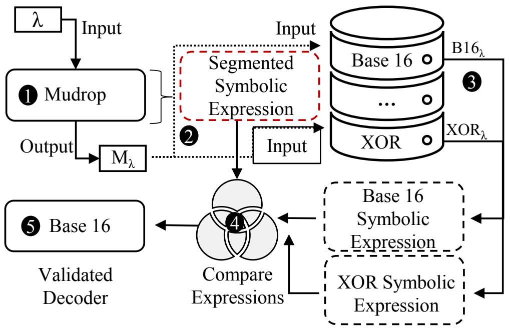


Figure 4: Symbolic Expression Matching For *Mudrop*.

D and the starting I_s and ending I_e boundary addresses used to segment the expression based on its attributes, i.e., addresses where the symbolic expressions grow. VADER parses only those parts of the expression within the boundary (Line 6 in Algorithm 2). Now, VADER compares the segmented expression with the symbolic reference implementation expression corresponding to D (Line 7 in Algorithm 2).

Compare Expressions. VADER 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 expressions, M_λ is used as input to decoding algorithms (Figure 4, ❷), ensuring the resulting expression (e.g., $B16_\lambda$ in Figure 4, 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, concrete values assumed during forking correspond to the previous constraints imposed by M_λ.

Now, having both symbolic expressions, VADER can compare them for equivalence (Figure 4, ❹). For comparison (Line 8 in Algorithm 2), there are two conditional constructs: (1) check whether the overall expressions match; if not, (2) solve for and compare the concrete 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 two versions of an XOR by `0x23` algorithm. As the expression is decoded, it grows with additional operations corresponding to algorithmic computations. For example, one byte of M_λ and XOR_λ is partially transformed into:

M_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, VADER invokes the symbolic solver to compare the concrete outputs of both expressions. Concrete values are based on the constraints from M_λ that are imposed on XOR_λ during its execution. This ensures that expressions are evaluated based on the same constraints, resulting in the same concrete output if they are functionally identical. Since each comparison is per explored path, the results are a ratio of matches versus non-matches, i.e., when concretized outputs for each path are equal, it is a match; otherwise, it is a non-match. If the ratio is high enough (explained in §4), their equivalence is confirmed (Figure 4, ❺).

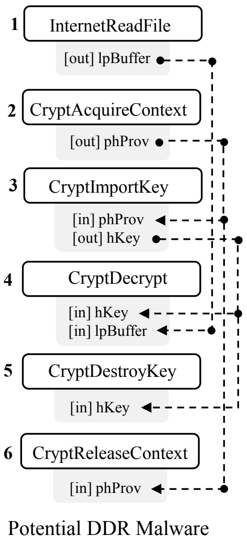


Figure 5: Constraint Chaining For Crypto Identification.

Compare Decryption Expressions. At this stage, VADER verifies decryption algorithms by partitioning boundaries based on source and sink markers. These boundaries are set by analyzing the constraints on sinks (*OBoundary*) in relation to their sources (*IBoundary*). It is important to recall that VADER employs taint analysis during DDR logic localization (§3.1). Figure 5 illustrates segmenting symbolic expressions with isolated boundaries (note: No discovered DDR malware sample encrypts dead drop content; this example uses a non-DDR malware sample to demonstrate the approach). The dotted lines represent the matching of input [in] constraints from a parameter with the output [out] constraints of at least one API predecessor. When a decryption API, like `CryptReleaseContext`, is used, VADER conducts a backward analysis of constraints to trace the API sequence, confirming the decryption algorithm. This *decryption constraint-chaining* method links the constraints of recovered APIs to their preceding APIs.

In Figure 5, decryption starts after data retrieval from the web app (1), followed by cryptographic context initialization (2) and key derivation (3) before data decryption (`lpBuffer`, 4). VADER uses backward analysis to identify decryption constraints and the sequence of APIs in the malware's symbolic expression, extracting only the relevant decryption portion.

To create reference symbolic expressions for comparison, VADER uses decryption models based on the API sequence from Table 2. By examining these implementations, VADER generates a symbolic expression and applies decryption constraint chaining to identify constraints related to the API sequence. The segmented expression is then compared with reference constraints using the *compareExpression* function (Algorithm 1, line 8). A match confirms that the decryption API computations align with the reference models. Upon matching, VADER produces a de-manipulation recipe detailing the decryption steps used by the DDR malware sample. This recipe can be applied to web app content to determine whether it represents a dead drop.

4 Pre-Deployment Evaluation

VADER is built upon S2E [22] for concolic analysis, with custom code for localizing DDR logic and identifying de-manipulation recipes. It integrates AVClass2, a widely utilized malware labeling tool [64, 54, 74, 49, 37, 112].

4.1 DDR Logic Localization

To evaluate VADER's ability to identify DDR logic, we tested it with five known DDR malware families, each with five randomly selected variants, totaling 25 ground truth samples. The performance evaluation is summarized in Table 3, which lists the web app, domain, malware family, and relevant reports detailing the ground truth, including the web app connection. Accuracy metrics such as true positive (TP), false positive (FP), and false negative (FN) values are also provided. Overall, VADER successfully localized the DDR logic in 23 out of 25 samples, achieving a 92% recall rate and an F1 score of 96%.

In our analysis, we identified two false FNs in *msil* (Table 3, Row 5). Manual review confirmed DDR malware, but VADER missed the dead drop link due to unresolved symbolic constraints, though this was rare. Additionally, VADER can analyze a DDR malware sample even when associated dead drops are inactive. For instance, in analyzing *comnie* (Row 2 of Table 3), the dead drop was inactive, yet VADER used concolic analysis to successfully localize the DDR logic. With only two FNs and a 92% recall in DDR logic localization, we are confident in VADER's ability to identify DDR malware and extract de-manipulation recipes for proactive dead drop discovery.

4.2 Validating Symbolic Expression Matching

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

Table 3: Validating DDR Logic Localization.

Web App	Domain	Family	Source	#	TP	FP	FN
Blockchain	blockchain.info	pony	[97]	5	5	0	0
Google Drive	drive.google.com	doina	[98]	5	5	0	0
Pastebin	kryptik	pastebin.com	[112]	5	5	0	0
Pastebin	msil	pastebin.com	[112]	5	3	0	2
X	x.com	razy	[6]	5	5	0	0
	Total		25	Acc. (92%)	23	0	2

de-manipulation algorithms in the same class consistently produce high-confidence matches, regardless of implementation details (i.e., BASE64 source code implementations).

Source Code Similarity. We selected up to three implementations of each algorithm listed in Table 2 from open-source repositories or libraries. Importantly, VADER is flexible and can be extended to support additional algorithm implementations. Using Moss [96], a widely used software similarity framework, we compared all implementations and found a 0% match between them, confirming their distinctiveness. For instance, no XOR implementation matches another, nor does it resemble implementations from other algorithm classes. Detailed results are provided in Appendix B.

Symbolic Expression Matching. Given the distinct source code implementations for each decoding algorithm category, we now need to compare the equivalence of symbolic expressions generated by VADER across different implementations of decoding algorithms. The validation result is shown in Table 4. With a symbolic input size of 8 bytes, the expression has 256^8 (4.2 billion) possible inputs. Ideally, we would evaluate the entire input space when comparing symbolic expressions, but this is time-prohibitive. Instead, we exhaust the input space for each pair over two hours to assess the number of matches in the output space. This is sufficient, as the ratio of non-matches to matches plateaus and stabilizes after 30 minutes within the two-hour window. A graphical representation of the plateau for each set of comparisons is presented in Appendix C.

As shown in Table 4, the symbolic expressions generated by VADER for different implementations within the same algorithm class achieve consistently high scores. This indicates that VADER can effectively recognize *functional equivalence* within algorithm classes. In contrast, expressions generated for algorithms across different classes show significantly lower scores, underscoring the differences in their operational logic. For example, Table 4c reveals that most comparisons (474/554[3]) between algorithms from different classes result in a 0% similarity match.

A specific case is observed in Table 4c, Row 7, where the symbolic expression for Base64 v1 shows low similarity scores when compared to those of other versions. Upon investigation, we determined that the match rates of 44% and 77% for Base64 v2 and v3, respectively, are attributable to extensive error-checking routines present in Base64 v1. These routines cause VADER to fork into two distinct paths during symbolic execution: (1) a success path and (2) a failure path that returns null. In contrast, Base64 v2 and v3 employ less stringent error-checking mechanisms, which

[3]71/625 algorithms are compared with themselves.

Table 4: Validation of Symbolic Expressions Produced By VADER To Assess Functional Equivalence Across De-Manipulation Algorithm Implementations.

(a) AES - RC4.

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

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

		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

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

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

result in symbolic expressions that predominantly reflect successful operations. This difference explains the observed lower similarity scores when v1 is compared to v2 and v3.

When comparing Base64 v2 and v3 to v1 (see Table 4c, Rows 8 and 9, Column 7), we observe match rates of 97%, 89%, and 100%, respectively. The difference is due to v2 and v3 lacking extensive error-checking routines. Since all versions process 8-byte input, if v2 completes symbolic execution first, it guides VADER along v1's success path, resulting in higher match rates. This highlights how error-checking differences impact symbolic exploration and similarity metrics.

Table 4c also shows a trend in Base decoding classes (e.g., Base16, Base32, Base64). All Base algorithms use lookup tables for character translation, creating baseline similarities due to shared structural elements. As a result, only six out of 144 comparisons have a 0% match, far fewer than the expected 108 when comparing unrelated Base classes. However, the noise in similarity scores for cross-class comparisons is still lower than in same-class comparisons. This ensures the observed similarities are not enough to confidently determine functional equivalence between unrelated Base algorithms.

Finally, our analysis demonstrates that VADER can identify de-manipulating algorithms, even with different implementations. For optimal performance in large-scale deployments, we select algorithm implementations with match rates over 90% within their class and under 25% across other classes (e.g., v1). This selection minimizes false positives and improves VADER 's reliability in identifying de-manipulation recipes.

Table 5: De-Manipulation Recipe Identification.

Web App	Family	Source	#	Algorithm	TP	FP	FN
Blockchain	pony	[97]	5	String Parsing, Base16	5	0	0
Google Drive	doina	[98]	5	None	0	5[1]	0
Pastebin	kryptik	[112]	5	Base64	5	0	0
Pastebin	msil	[112]	5	Base64	5	0	0
X	razy	[6]	5	String Parsing, Base64	5	0	0
None	spora	[60]	5	AES	4	0	1
Total			30	Acc. (83%)	24	5	1

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

4.3 De-Manipulation Recipe Identification

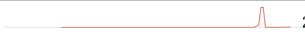
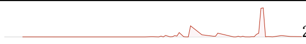
Table 5 summarizes our evaluation of IO boundary isolation and de-manipulation algorithm verification. In all cases, the IO boundary aligned with the verified algorithm, with no instances of multiple algorithms per boundary. Columns 1 and 2 provide details on the web app and malware family. For validation, we expanded our dataset to include *spora*, which includes cryptographic routines. Column 4 lists the number of variants per malware family, and Column 5 shows the de-manipulation recipes. The final columns present accuracy metrics: TP, FP, and FN. Overall, VADER identified recipes for 30 malware samples, achieving a 96% recall rate and an F1 score of 89%.

We also found five FPs. For instance, *doina* requests a Google Drive file with a plaintext IP, but VADER flagged it as String Parsing due to format validation resembling parsing. Since the content is plaintext, these FPs have minimal impact on proactive dead drop discovery. VADER also struggles with unresolved symbolic constraints, which limits constraint chaining for source-to-sink mapping. Despite these issues, the low FP and FN highlight VADER 's accuracy in identifying de-manipulation recipes.

5 Post-Deployment Findings

We deployed VADER on a 100k malware dataset built by randomly downloading Windows executables uploaded to VirusTotal between 2012 and 2022 [104]. Each malware sample triggered more than one anti-virus (AV) engine, based on dataset parameters from Zhu et

Table 6: Distribution Of DDR Malware Found In Our Study.

Web App	Domain	# M[1]	Dead Drop	Malware/Year 2012 - 2022	First Seen	Last Seen	# F[1]	Backup Dead Drop	Total Recipes	Unique Recipes
Pastebin	pastebin.com	6,053	44		2014-02	2022-06	32	0	6,053	3
Blockchain	blockchain.info	776	7		2017-12	2019-11	59	111	776	1
	blockcypher.com	573	23		2017-12	2019-11	41	415	573	1
	bitaps.com	617	4		2017-11	2021-10	16	423	617	1
	coinmarketcap.com	45	1		2017-12	2019-11	10	45	45	1
	blockr.io	200	14		2017-12	2021-06	5	200	200	1
Blockchain Aggregate		2,200	37[2]		2017-11	2021-06	81[2]	1,194	2,200	1
X	twitter.com	473	11		2017-10	2022-03	8	3	473	2
Google	docs.google.com	9	1		2018-02	2022-12	1	1	9	1
	googleusercontent.com	137	115		2017-06	2020-06	8	0	137	1
	drive.google.com	9	1		2018-02	2022-12	1	1	9	1
Dropbox	dropbox.com	11	2		2017-06	2020-06	2	0	11	1
GitHub	github.com	7	2		2017-11	2018-10	2	0	7	1
WordPress	wordpress.com	4	1		2012-09	2014-08	1	0	4	1
Total		8,906	200		2012-09	2022-12	110[2]	1,199	8,906	7[2]

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

al. [117], who studied VirusTotal labeling dynamics. Tranco [62] ensured the malware connected to at least one web app.

We present the distribution of DDR malware in our study, focusing on web apps used by DDR malware in Table 6. For practical application in intrusion detection, we applied VADER 's analysis methods to real-world web app network traffic from Netresec [83] (Table 7), cited in top-tier research [68, 52]. For HTTPS traffic, we accessed the web app endpoint, decoded the response with the recipe, and extracted IP/URL addresses, which we scanned on VirusTotal [104] and URLhaus [1] to confirm maliciousness. Finally, we provide two case studies to demonstrate the prevalence and complexity of de-manipulation recipes in malware and a practical approach to proactively discover dead drops.

5.1 DDR-Enabling Web Apps

Table 6 shows the distribution of DDR malware. Column 1 lists web apps, Column 2 their domains, and Column 3 the number of malware samples (#M) using each app. Column 4 shows unique dead drops, while Columns 5–7 provide a temporal snapshot over 11 years, using first- and last-seen dates from VirusTotal. Column 8 reports distinct malware families (#F), and Column 9 highlights backup dead drop domains, used if the primary fails. Columns 10–11 list de-manipulation recipe counts and unique recipes discovered.

From Table 6, Pastebin accounts for about 68% of discovered DDR malware (6,053). VADER found 44 unique dead drops (22% of all) across 32 malware families. The caes of Pastebin DDR malware that we identified span 2014-2022, with a major spike in 2021-2022, accounting for nearly 5k discoveries. Historically, Pastebin has hosted stolen content or malware [36]. This work is among the first to expose its pervasiveness as a platform for hosting dead drops. VADER identified one recipe per malware and four unique recipes for Pastebin DDR malware (Columns 10–11). The low count of unique recipes relative to malware volume shows VADER is scalable for proactive detection.

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

VADER also found 11 X.com dead drops in 473 malware. We classified dead drop origins as (1) hard-coded, like traditional DDR, and (2) dynamically generated, similar to DGA malware. This emerged in the sample *fugrafa*, which uses X.com to host encoded C&C addresses. VADER found three backup X.com dead drops during DDR logic localization. Further analysis showed that the account name varies based on a function's output, generating accounts like `np8j7ovqdl`, `q5euqysfu5`, and `qistp743li`. Despite this complexity, VADER, combined with DGA counteraction techniques, can effectively mitigate such threats [5, 61].

VADER identified 117 Google dead drops, where data on Google web apps is accessed via unique file IDs. By analyzing file URLs, we traced the endusers for each file on Google Drive and Docs, finding both were hosted by the same user. VADER also discovered dead drops on Dropbox and GitHub, with 11 and seven malware instances, respectively. One WordPress dead drop and its de-manipulation recipe were included in our findings, along with three more WordPress dead drops uncovered through proactive discovery (see §2).

Row *Total*, Column 4 reveals spikes in DDR adoption from 2012–2022, showing a cycle of growth, delayed remediation, and resurgence. This trend, especially the recent rise, highlights the need for proactive discovery to curb the reemergence of DDR malware, particularly as new strains appear.

In total, VADER identified 8,906 samples and 200[4], including 1,199 with backups. VADER also uncovered seven unique de-manipulation recipes, demonstrating the scalability of our approach. We are collaborating with web app providers on remediation and have disrupted at least 6,674 DDR samples by removing their dead drops.

5.2 VADER-Enabled Proactive Discovery

The rise of DDR malware underscores the need for proactive measures to uncover hidden dead drops. Using de-manipulation recipes from DDR malware samples, we applied them to the Netresec [83] network traffic dataset, summarized in Table 7. Column 1 lists the recipes used to decode web app responses. After decoding, we cross-referenced the data with URLhaus [1] and VirusTotal [104] to identify potential C&C addresses, recording infected web apps and dead drops in Columns 2 and 3. Column 4 shows unique C&C addresses, while Column 5 provides the first submission date (FSD) from URLhaus, correlating with the C&C server discovery date. Column 6 reports active C&C addresses. To determine C&C server liveness, domain-based addresses were checked for website availability, and IP-based addresses were verified using `nmap` to confirm open ports, following a less intrusive method from prior research [37].

To evaluate VADER in real-world scenarios, we assessed false positives (FPs) and false negatives (FNs), as shown in Columns 7 and 8. An FP occurs when a dead drop candidate is later found to serve benign purposes. Since many web apps (e.g., OneDrive, Dropbox, Discord) do not disclose user activity details (e.g., login times, IP addresses), determining intent behind a dead drop is difficult. Therefore, we rely on service providers for verification, marking a dead drop as an FP if it remains active after notification. This helps estimate the upper bound of FPs. FNs occur when a dead drop was created before its C&C address was added to the blocklist, reflecting VADER's ability to detect dead drops quickly. For platforms that do not disclose content creation times (e.g., OneDrive), these entries are marked N/A in Column 8.

From Table 6, VADER identified seven web apps hosting dead drops. Recipe-enabled proactive discovery, shown in Column 2 of Table 7, uncovered 11 web apps, a 57.1% increase. The use of de-manipulation recipes identified 72 dead drops, averaging 6.5 per web app (Column 3), highlighting the significant role of recipe identification in uncovering threats. Notably, only two recipes involved a single de-manipulation layer (Base64 and String Parsing). With over 40% of dead drops using multiple layers, brute-force approaches are infeasible, emphasizing VADER 's effectiveness.

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

[4]Analyzing 100k malware samples revealed 273 dead drops (see Appendix D).

Table 7: Applying Recipes On Network Traffic Dataset To Proactively Discover Dead Drops

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

1: First Submission Date on URLhaus [1]. 2: Number of live C&C servers.
3: A false positive (FP) is reported if a proactively identified dead drop is not taken down by web app providers.
4: A false negative (FN) is reported if the creation time of the dead drop is earlier than the time the C&C address is added to the blocklist.

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

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

5.3 Case Studies

Our first case study demonstrates the prevalence and complexity of de-manipulation recipes. We also show that VADER effectively detects C&C addresses on Pastebin, prevalent in our research. While our access to Pastebin is limited, this highlights the advantages of VADER, especially for web app providers with full platform access.

5.3.1 Prevalence and Complexity. To understand the prevalence and complexity of identified recipes in malware, we analyze the malware in this study. Table 8 shows de-manipulation recipes in Column 1 with their prevalence over time in Column 2. Columns 3 and 4 count malware and families, while Columns 4 and 5 list associated web apps and unique dead drops. Column 6 displays

Table 8: De-Manipulation Recipes Identified.

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

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

content types, and the final columns show algorithmic complexity in basic blocks.

Among the nine decoders listed in Table 2, VADER identified five, with XOR being the only cryptographic algorithm used by DDR malware in our dataset. This aligns with observed trends [37, 94], where encryption is often avoided due to its complexity. In contrast, decoding algorithms are platform-independent and widely supported, allowing malware authors to reuse routines across systems and environments.

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

Base64 typically encodes binary data into ASCII for C&C URLs, while Base16 encodes IPs in hex to obscure them. As shown in Table 1, the *mudrop* malware uses Base16 decoding to extract its C&C IP from dead drop content. Other examples include Base16+Chr Rot (Row 5) and Base16+Str Prs+Str⟶Int (Row 6), the latter being specific to blockchain web apps. However, since the `netresec` dataset in §5.2 lacks blockchain connections, this technique wasn't detected.

The *Recipe Complexity* columns reflect how challenging it is to generate de-manipulation recipes. Malware using three or more algorithms may span over 69 basic blocks (Row 6). Even simpler cases, like Str Prs, which extracts and parses dead drop content, can reach up to 27 basic blocks. On average, each sample uses 1.5 de-manipulation algorithms per recipe. Despite this, VADER effectively generates these recipes. We have shown how they enable discovering dead drops from network traffic and will next demonstrate their use in analyzing web app content.

5.3.2 Proactive Discovery Via Web App Scanning. *Banload* uses Pastebin at `http://pastebin.com/raw/jYzmPCEr` as its dead drop. VADER identified `c2t5cGUubXlkZG5zLm1l` and the Base64 de-manipulation recipe (Rows 1 and 2 in Table 9). Applying this recipe, VADER decoded the C&C address `skype.myddns.me` and can use this de-manipulation recipe to proactively discover previously undisclosed dead drops.

With VADER, data is randomly extracted from Pastebin using eight-character labels, de-manipulated via the recipe, and parsed with regex for IPs or URLs. These addresses are checked against URLhaus or VirusTotal for malicious activity, and associated accounts are flagged for remediation. As noted in §2.3, web app providers can scan platforms at scale for abuse, demonstrating VADER's potential for proactive dead drop discovery.

In Table 9, we present previously undisclosed dead drops and C&C addresses. Column 1 lists the discovered Pastebin dead drops, Column 2 shows the dead drop content, and Column 3 provides the associated C&C addresses. Our proactive approach uncovered 13 previously unknown dead drops, with seven linked to the same

Table 9: VADER's Banload Analysis.

VADER's *Banload* Analysis Output	
Web App Account	`http://pastebin.com/raw/jYzmPCEr`
↳Dead Drop Content	`c2t5cGUubXlkZG5zLm1l`
↳Recipe	`Base64 Decoding`
↳C&C Address	`skype.myddns.me`

Proactive Discovery: Pastebin		
Dead Drop	**Content**	**C&C Address**
`0hx614sm`	`c2t5cGUubXlkZG5zLm1l`	`skype.myddns.me`
`4yAyQVev`	`d3d3LnRvcDE2OC5vcmc=`	`www.top168.org`
`AHtqg5QQ`	`c2t5cGUubXlkZG5zLm1l`	`skype.myddns.me`
`FaZqCHMN`	`c2t5cGUubXlkZG5zLm1l`	`skype.myddns.me`
`grHusy2R`	`c2t5cGUubXlkZG5zLm1l`	`skype.myddns.me`
`L2QEYA3P`	`d3d3LnRvcDE2OC5vcmc=`	`www.top168.org`
`V7Cm4Vid`	`c2t5cGUubXlkZG5zLm1l`	`skype.myddns.me`
`Vr8Bequa`	`c2t5cGUubXlkZG5zLm1l`	`skype.myddns.me`
`XcHYKsYP`	`c2t5cGUubXlkZG5zLm1l`	`skype.myddns.me`
`WpU6vm5L`	`cHJvZmFzc2lzdGFuY2UuY29t`	`profassistance.com`
`xj8s4nF1`	`MTIxLjIyNi4yMzcuNQ==`	`121.226.237.5`
`EALLXQZw`	`MTEwLjE4Mi43OC4xNg==`	`103.254.75.18`
`DdEhn3D7`	`ZnVlbHJlc2N1ZS5pZQ==`	`fuelrescue.ie`

C&C address as *banload*. Of the six remaining C&C addresses, five were unique. These results highlight that, even with limited data extraction resources compared to web app providers, VADER combined with internal tools can already enhance platform security and deliver proactive outcomes.

6 Related Work

Malware-Driven Web Application Abuse. Existing research has investigated web apps abused for botnet activities, primarily focusing on end-user perspectives [8, 67, 86, 102, 3, 69, 116, 114, 27]. Recent efforts have aimed to examine the intricate relationship between web apps and the malware that leverages them for victimization [112]. While studies have targeted specific malware families [29, 97, 91, 93, 30, 98] and the blockchain abuse [40, 41, 89, 45, 113], their findings largely propose *reactive* countermeasures. Before VADER, *proactively* discovering dead drops remained largely unaddressed.

Data Manipulation Identification. Prior research has focused on binary code extraction and function identification using dynamic or symbolic analysis [14, 53, 95, 79, 108, 26, 70, 18, 88, 106, 65, 72, 63, 28]. These methods are limited by code coverage or reliance on available network endpoints. In contrast, VADER 's concolic analysis improves malware exploration even with dead endpoints. ByteWeight [9] uses deep neural networks to identify de-manipulation algorithms but needs a large training dataset. VADER, however, uses symbolic expression matching to identify de-manipulation algorithms regardless of implementation.

Leveraging Concolic Analysis. Concolic analysis has been widely used by the research community in bug identification [15, 21, 17], test case generation [51, 13, 23], dynamic analysis [87, 50, 80, 110, 2], and taint analysis [92, 48, 84, 24]. However, since these related works are largely geared towards vulnerability analysis, they are not easily extensible to help web app providers mitigate DDR practices. In contrast, VADER is among the first proposals to help web app providers confirm DDR capability from malware samples and extract de-manipulation recipes. This capability enables web app providers to proactively identify dead drops.

C&C Address Identification. Existing techniques detect C&C addresses by analyzing malicious network traffic [42, 11, 111, 105, 47, 100], but these typically focus on identifying C&C addresses for individual malware samples. In contrast, VADER identifies multiple dead drops from a single malware sample, which may conceal different C&C addresses. While tools like Cyberprobe [81] and ExecScent [82] probe remote endpoints to detect C&C addresses, they are ineffective against DDR malware that hides C&C addresses behind dead drops. Instead, VADER can uncover previously undisclosed dead drops, enabling web app providers to address abuse and counteract botnets.

7 Discussion

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

Evasion Techniques. While malware authors often use evasion techniques like probing the execution environment to avoid analysis, VADER addresses many of these strategies by hooking system APIs (see §3.1). Although comprehensive evasion mitigation is out of scope of this paper, VADER models commonly observed evasive APIs identified by industry and academic experts [7, 55, 71, 38]. For example, when `GetLocaleInfo` is invoked to detect the bot's location, VADER injects symbolic return values to trick the DDR malware sample into continued execution. This enables effective analysis even in the presence of evasive behavior. Our implementation, evaluated on a malware sample set collected between 2012 and 2022, achieves 92% accuracy in localizing the DDR logic. Thanks to the extensible design of VADER's, web app providers can add support for emerging evasion techniques. A full list of modeled evasive APIs is available in Appendix A.

Advanced Packers and Obfuscators. Although advanced packers and obfuscators were not observed in our dataset, advanced packers (e.g., VMProtect) or obfuscators (e.g., O-LLVM) can be deployed, complicating DDR malware detection and recipe extraction for web app providers. These challenges are not unique to VADER, but reflect broader issues in symbolic analysis, where handling such techniques often requires added human effort. Several approaches have been proposed [12, 109, 16, 115] to recover the semantics of obfuscated code. As VADER does not rely on any special functionality in S2E, web app providers can readily extend it to support advanced packers and obfuscators in future DDR malware.

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

8 Conclusion

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

Acknowledgment

We thank the anonymous reviewers for their constructive comments and feedback. We also thank our collaborators at Netskope for their support, insights, and suggestions throughout this research. This material was supported in part by the Office of Naval Research (ONR) under grants N00014-19-1-2179 and N00014-23-1-2073; the National Science Foundation (NSF) under grant 2143689; and the Defense Advanced Research Projects Agency (DARPA) under contract N66001-21-C-4024. Any opinions, findings, and conclusions in this paper are those of the authors and do not necessarily reflect the views of our sponsors, collaborators, the United States Military Academy, the Department of the Army, or the Department of Defense.

References

[1] Abuse.ch. 2023. URLHaus. (2023). Retrieved March 12, 2023 from https://urlhaus.abuse.ch/.

[2] Omar Alrawi, Moses Ike, Matthew Pruett, Ranjita Pai Kasturi, Srimanta Barua, Taleb Hirani, Brennan Hill, and Brendan Saltaformaggio. 2021. Forecasting Malware Capabilities From Cyber Attack Memory Images. In *Proc. 30th USENIX Security.* (Aug. 2021).

[3] Sumayah Alrwais, Kan Yuan, Eihal Alowaisheq, Zhou Li, and XiaoFeng Wang. 2014. Understanding the dark side of domain parking. In *Proc. 23rd USENIX Security.* (Aug. 2014).

[4] Manos Antonakakis, Roberto Perdisci, Wenke Lee, Nikolaos Vasiloglou II, and David Dagon. 2011. Detecting Malware Domains at the Upper DNS Hierarchy. In *Proc. 20th USENIX Security.* (Aug. 2011).

[5] Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. 2012. From Throw-away Traffic to Bots: Detecting the Rise of DGA-based Malware. In *Proc. 21st USENIX Security.* (Aug. 2012).

[6] Pieter Arntz. 2017. Analyzing malware by API calls. (Oct. 2017). Retrieved March 06, 2024 from https://blog.malwarebytes.com/threat-analysis/2017/10/analyzing-malware-by-api-calls/.

[7] MITRE | ATT&CK. 2021. Attack Matrix for Enterprise. (Nov. 2021). Retrieved November 06, 2021 from https://attack.mitre.org/.

[8] Hammi Badis, Guillaume Doyen, and Rida Khatoun. 2014. Understanding Botclouds From a System Perspective: A Principal Component Analysis. In *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS).* IEEE, (May 2014).

[9] Tiffany Bao, Jonathan Burket, Maverick Woo, Rafael Turner, and David Brumley. 2014. BYTEWEIGHT: Learning to Recognize Functions in Binary Code. In *Proc. 23rd USENIX Security.* (Aug. 2014).

[10] Lenart Bermejo and Joelson Soares. 2018. Lazarus Targets Latin American Financial Companies. (Nov. 2018). Retrieved November 15, 2024 from https://www.trendmicro.com/en_us/research/18/k/lazarus-continues-heists-mounts-attacks-on-financial-organizations-in-latin-america.html.

[11] Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. 2012. Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In *Proc. 28th Annual Computer Security Applications Conference (ACSAC).*

[12] Tim Blazytko, Moritz Contag, Cornelius Aschermann, and Thorsten Holz. 2017. Syntia: synthesizing the semantics of obfuscated code. In *Proc. 26th USENIX Security.* (Aug. 2017).

[13] Robert S Boyer, Bernard Elspas, and Karl N Levitt. 1975. SELECT — A Formal System for Testing and Debugging Programs by Symbolic Execution. In ACM.

[14] Juan Caballero, Noah M Johnson, Stephen McCamant, Dawn Song, and UC Berkeley. 2010. Binary Code Extraction and Interface Identification for Security Applications. In *Proc. 17th NDSS.* (Feb. 2010).

[15] Cristian Cadar and Dawson Engler. 2005. Execution Generated Test Cases: How to Make Systems Code Crash Itself. In *Proc. International SPIN Workshop on Model Checking of Software.* Springer. (Aug. 2005).

[16] Joan Calvet, José M. Fernandez, and Jean-Yves Marion. 2012. Aligot: cryptographic function identification in obfuscated binary programs. In *Proc. 19th ACM CCS.* (Oct. 2012).

[17] Sang Kil Cha, Thanassis Avgerinos, Alexandre Rebert, and David Brumley. 2012. Unleashing Mayhem on Binary Code. In *Proc. 33rd IEEE Security and Privacy.* (May 2012).

[18] Mahinthan Chandramohan, Yinxing Xue, Zhengzi Xu, Yang Liu, Chia Yuan Cho, and Hee Beng Kuan Tan. 2016. Bingo: Cross-Architecture Cross-OS Binary Search. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering.* (Nov. 2016).

[19] Joseph C Chen, Kenney Lu, Jaromir Horejsi, and Gloria Chen. 2021. Biopass RAT: New Malware Sniffs Victims via Live Streaming. (July 2021). Retrieved November 06, 2024 from https://www.trendmicro.com/en_us/research/21/g/biopass-rat-new-malware-sniffs-victims-via-live-streaming.html.

[20] Anton Cherepanov. 2017. Analysis of TeleBots' cunning backdoor. (July 2017). Retrieved November 15, 2022 from https://www.welivesecurity.com/2017/07/04/analysis-of-telebots-cunning-backdoor/.

[21] Vitaly Chipounov, Vlad Georgescu, Cristian Zamfir, and George Candea. 2009. Selective Symbolic Execution. In *Proc. 5th Workshop on Hot Topics in System Dependability (HotDep).* (June 2009).

[22] Vitaly Chipounov, Volodymyr Kuznetsov, and George Candea. 2011. S2E: A Platform for In-vivo Multi-path Analysis of Software Systems. *ACM SigPlan Notices.*

[23] Lori A. Clarke. 1976. A System to Generate Test Data and Symbolically Execute Programs. *IEEE Transactions on Software Engineering*, 3, 215–222.

[24] James Clause, Wanchun Li, and Alessandro Orso. 2007. Dytan: A Generic Dynamic Taint Analysis Framework. In *Proc. International Symposium on Software Testing and Analysis (ISSTA).* (July 2007).

[25] CYFIRMA. 2022. Cyber Research on the Malicious Use of Discord. (Sept. 2022). Retrieved March 12, 2024 from https://www.cyfirma.com/research/cyber-research-on-the-malicious-use-of-discord.

[26] Yaniv David, Nimrod Partush, and Eran Yahav. 2016. Statistical Similarity of Binaries, (June 2016).

[27] Ruian Duan, Omar Alrawi, Ranjita Pai Kasturi, Ryan Elder, Brendan Saltaformaggio, and Wenke Lee. 2021. Towards measuring supply chain attacks on package managers for interpreted languages. In *Proc. 2021 NDSS.* (Feb. 2021).

[28] Manuel Egele, Maverick Woo, Peter Chapman, and David Brumley. 2014. Blanket execution: dynamic similarity testing for program binaries and components. In *Proc. 23rd USENIX Security.* (Aug. 2014).

[29] ESET. 2019. Operation Ghost. The Dukes aren't back — they never left. (Oct. 2019). Retrieved February 26, 2022 from https://www.welivesecurity.com/wp-content/uploads/2019/10/ESET_Operation_Ghost_Dukes.pdf.

[30] FireEye. 2019. APT17: Hiding in Plain Sight - FireEye and Microsoft Expose Obfuscation Tactic. (Feb. 2019). Retrieved Feburary 21, 2021 from https://www.fireeye.com/current-threats/apt-groups/rpt-apt17.html.

[31] FireEye. 2019. Double Dragon - APT41, a Dual Espionage and Cyber Crime Operation. (Aug. 2019). Retrieved March 06, 2022 from https://content.fireeye.com/apt-41/rpt-apt41.

[32] FireEye. 2016. Multigrain - Point of Sale Attackers Make an Unhealthy Addition to the Pantry. (Apr. 2016). Retrieved November 6, 2021 from https://www.fireeye.com/blog/threat-research/2016/04/multigrain_pointo.html.

[33] FIRST. 2015. FIRST is the global Forum of Incident Response and Security Teams. (Oct. 2015). Retrieved December 31, 2024 from https://www.first.org.

[34] Fraunhofer FKIE. 2021. Malpedia: Free and Open Malware Reverse Engineering Resource offered by Fraunhofer FKIE. (Nov. 2021). Retrieved November 6, 2021 from https://malpedia.caad.fkie.fraunhofer.de.

[35] Forcepoint. 2022. Monsoon – Analysis of an APT Campaign. (Mar. 2022). Retrieved March 6, 2022 from https://www.forcepoint.com/sites/default/files/resources/files/forcepoint-security-labs-monsoon-analysis-report.pdf.

[36] Fortinet. 2019. The Malicious Use of Pastebin. (2019). Retrieved March 12, 2022 from https://www.fortinet.com/blog/threat-research/malicious-use-of-pastebin.

[37] Jonathan Fuller, Ranjita Pai Kasturi, Amit Sikder, Haichuan Xu, Berat Arik, Vivek Verma, Ehsan Asdar, and Brendan Saltaformaggio. 2021. C3PO: Large-Scale Study of Covert Monitoring of C&C Servers via Over-Permissioned Protocol Infiltration. In *Proc. 28th ACM CCS.* (Nov. 2021).

[38] Nicola Galloro, Mario Polino, Michele Carminati, Andrea Continella, and Stefano Zanero. 2022. A Systematical and Longitudinal Study of Evasive Behaviors in Windows Malware. *COSE.*

[39] Ibrahim Ghafir, Vaclav Prenosil, Mohammad Hammoudeh, Thar Baker, Sohail Jabbar, Shehzad Khalid, and Sardar F. Jaf. 2018. Botdet: a system for real time botnet command and control traffic detection. *IEEE Access*, 6, 38947–38958.

[40] Mar Gimenez-Aguilar, Jose Maria de Fuentes, and Lorena Gonzalez-Manzano. 2023. Malicious uses of Blockchains by Malware: From the Analysis to Smart-Zephyrus. *International Journal of Information Security*, 1–36.

[41] Gibran Gomez, Pedro Moreno-Sanchez, and Juan Caballero. 2022. Watch Your Back: Identifying Cybercrime Financial Relationships in Bitcoin through Back-and-Forth Exploration. In *Proc. 29th ACM CCS.* (Nov. 2022), 1291–1305.

[42] Guofei Gu, Phillip A Porras, Vinod Yegneswaran, Martin W Fong, and Wenke Lee. 2007. Bothunter: detecting malware infection through ids-driven dialog correlation. In *Proc. 16th USENIX Security.* (Aug. 2007).

[43] Guofei Gu, Junjie Zhang, and Wenke Lee. 2008. Botsniffer: detecting botnet command and control channels in network traffic. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2008, San Diego, California, USA, 10th February - 13th February 2008.* (Feb. 2008).

[44] Zuzana Hromcová and Anton Cherepanov. 2020. Invisimole: The Hidden Part of the Story Unearthing Invisimole's Espionage Toolset and Strategic Cooperations. (June 2020). Retrieved November 15, 2024 from https://www.welivesecurity.com/wp-content/uploads/2020/06/ESET_InvisiMole.pdf.

[45] Zhangrong Huang, Ji Huang, and Tianning Zang. 2020. Leopard: Understanding the Threat of Blockchain Domain Name Based Malware. In *Proceddings of the 21st International Conference Of Passive and Active Measurement (PAM).* (Mar. 2020).

[46] The Intercept. 2024. The Feds Are Coming For "Extremist" Gamers. (Mar. 2024). Retrieved June 9, 2024 from https://theintercept.com/2024/03/09/fbi-dhs-gamers-extremism-violence/.

[47] Gregoire Jacob, Ralf Hund, Christopher Kruegel, and Thorsten Holz. 2011. Jackstraws: Picking Command and Control Connections from Bot Traffic. In *Proc. 20th USENIX Security.* (Aug. 2011).

[48] Min Gyung Kang, Stephen McCamant, Pongsin Poosankam, and Dawn Song. 2011. Dta++: Dynamic Taint Analysis With Targeted Control-Flow Propagation. In *Proc. 18th NDSS.* (Feb. 2011).

[49] Doowon Kim, Bum Jun Kwon, Kristián Kozák, Christopher Gates, and Tudor Dumitraș. 2018. The Broken Shield: Measuring Revocation Effectiveness in the Windows Code-Signing PKI. In *Proc. 27th USENIX Security.* (Aug. 2018).

[50] Kyungtae Kim, I Luk Kim, Chung Hwan Kim, Yonghwi Kwon, Yunhui Zheng, Xiangyu Zhang, and Dongyan Xu. 2017. J-force: Forced Execution on Javascript. In *Proc. 26th International World Wide Web Conference (WWW).* (Apr. 2017).

[51] James C King. 1976. Symbolic Execution and Program Testing. In number 7. Vol. 19. ACM, 385–394.

[52] Stephan Kleber and Frank Kargl. 2011. Poster: network message field type recognition. In *Proc. 26th ACM CCS.* (Nov. 2011).

[53] Clemens Kolbitsch, Thorsten Holz, Christopher Kruegel, and Engin Kirda. 2010. Inspector Gadget: Automated Extraction of Proprietary Gadgets from Malware Binaries. In *Proc. 31th IEEE Security and Privacy.* (May 2010).

[54] Platon Kotzias, Leyla Bilge, and Juan Caballero. 2016. Measuring PUP Prevalence and PUP Distribution through Pay-Per-Install Services. In *Proc. 25th USENIX Security.* (Aug. 2016).

[55] Alexander Küchler, Alessandro Mantovani, Yufei Han, Leyla Bilge, and Davide Balzarotti. 2021. Does Every Second Count? Time-Based Evolution of Malware Behavior in Sandboxes. In *Proc. 2021 NDSS.* (Feb. 2021).

[56] CyFI Lab. 2025. Baseline Comparison For De-Manipulation Algorithm C/C++ Source Code Via Moss. (Apr. 2025). Retrieved April 09, 2025 from https://github.com/CyFI-Lab-Public/VADER/blob/main/appendix/moss.pdf.

[57] CyFI Lab. 2025. Baseline Comparison For Decoding Algorithm Similarity Via Symbolic Expressions. (Apr. 2025). Retrieved April 09, 2025 from https://github.com/CyFI-Lab-Public/VADER/blob/main/appendix/baseline_comparison.pdf.

[58] CyFI Lab. 2025. Defensive Evasion APIs. (Apr. 2025). Retrieved April 09, 2025 from https://github.com/CyFI-Lab-Public/VADER/blob/main/appendix/defensive_evasion_apis.pdf.

[59] CyFI Lab. 2025. Identified Dead Drops. (Apr. 2025). Retrieved April 09, 2025 from https://github.com/CyFI-Lab-Public/VADER/blob/main/appendix/identified_accounts.pdf.

[60] Malwarebytes Labs. 2017. Explained: Spora ransomware. (Mar. 2017). Retrieved November 15, 2022 from https://www.malwarebytes.com/blog/news/2017/03/spora-ransomware.

[61] Victor Le Pochat, Sourena Maroofi, Tom Van Goethem, Davy Preuveneers, Andrzej Duda, Wouter Joosen, Maciej Korczyński, et al. 2020. A Practical Approach for Taking Down Avalanche Botnets Under Real-World Constraints. In *Proc. 2020 NDSS.* (Feb. 2020).

[62] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. 2019. Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. In *Proc. 2019 NDSS.* (Feb. 2019).

[63] Pierre Lestringant, Frédéric Guihéry, and Pierre-Alain Fouque. 2015. Automated Identification of Cryptographic Primitives in Binary Code with Data Flow Graph Isomorphism. In *Proc. 10th ACM Symposium on Information, Computer and Communications Security (ASIACCS).* (Apr. 2015).

[64] Chaz Lever, Platon Kotzias, Davide Balzarotti, Juan Caballero, and Manos Antonakakis. 2017. A Lustrum of Malware Network Communication: Evolution and Insights. In *Proc. 38th IEEE Security and Privacy.* (May 2017).

[65] Juanru Li, Zhiqiang Lin, Juan Caballero, Yuanyuan Zhang, and Dawu Gu. 2018. K-Hunt: Pinpointing Insecure Cryptographic Keys from Execution Traces. In *Proc. 25th ACM CCS.* (Oct. 2018).

[66] Xiaojing Liao, Sumayah A. Alrwais, Kan Yuan, Luyi Xing, XiaoFeng Wang, Shuang Hao, and Raheem A. Beyah. 2016. Lurking malice in the cloud: understanding and detecting cloud repository as a malicious service. In *Proc. 23rd ACM CCS.* (Oct. 2016).

[67] Greeshma Lingam, Rashmi Ranjan Rout, Durvasula V. L. N. Somayajulu, and Sajal K. Das. 2020. Social Botnet Community Detection: A Novel Approach based on Behavioral Similarity in Twitter Network using Deep Learning. In *Proc. 15th ACM Symposium on Information, Computer and Communications Security (ASIACCS).* (Oct. 2020).

[68] Zhiheng Liu, Zhen Zhang, Yinzhi Cao, Zhaohan Xi, Shihao Jing, and Humberto La Roche. 2018. Towards a secure zero-rating framework with three parties. In *Proc. 27th USENIX Security.* (Aug. 2018).

[69] Wei Lu, Mark Miller, and Ling Xue. 2017. Detecting Command and Control Channel of Botnets in Cloud. In *Proceedings of the 1st International Conference on Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments (ISDDC).*

[70] Lannan Luo, Jiang Ming, Dinghao Wu, Peng Liu, and Sencun Zhu. 2014. PSemantics-Based Obfuscation-Resilient Binary Code Similarity Comparison With Applications to Software Plagiarism Detection. In *Proc. 22nd ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE).* (Nov. 2014).

[71] Lorenzo Maffia, Dario Nisi, Platon Kotzias, Giovanni Lagorio, Simone Aonzo, and Davide Balzarotti. 2021. Longitudinal Study of the Prevalence of Malware Evasive Techniques. *arXiv preprint arXiv:2112.11289.*

[72] Carlo Meijer, Veelasha Moonsamy, and Jos Wetzels. 2021. Where's Crypto?: Automated Identification and Classification of Proprietary Cryptographic Primitives in Binary Code. In *Proc. 30th USENIX Security.* (Aug. 2021).

[73] Joseph Menn. 2020. Court orders seizure of ransomware botnet controls as U.S. election nears. (Sept. 2020). Retrieved July 18, 2022 from https://www.reuters.com/article/us-uselection-cyber-botnet/court-orders-seizure-of-ransomware-botnet-controls-as-u-s-election-nears-idUSKBN26X1G2.

[74] Xianghang Mi, Xuan Feng, Xiaojing Liao, Baojun Liu, XiaoFeng Wang, Feng Qian, Zhou Li, Sumayah Alrwais, Limin Sun, and Ying Liu. 2019. Resident Evil: Understanding Residential IP Proxy as a Dark Service. In *Proc. 40th IEEE Security and Privacy.* (May 2019).

[75] Trend Micro. 2021. Malware Campaign Targets South Korean Banks. (Nov. 2021). Retrieved November 6, 2021 from https://blog.trendmicro.com/trendlabs-security-intelligence/malware-campaign-targets-south-korean-banks-uses-pinterest-as-cc-channel/.

[76] Trend Micro. 2022. Operation Endtrade: Tick's Multi-Stage Backdoors for Attacking Industries and Stealing Classified Data. (Nov. 2022). Retrieved November 15, 2022 from https://documents.trendmicro.com/assets/pdf/Operation-ENDTRADE-TICK-s-Multi-Stage-Backdoors-for-Attacking-Industries-and-Stealing-Classified-Data.pdf.

[77] Trend Micro. 2024. Trend Micro Collaborated with Interpol in Cracking Down Grandoreiro Banking Trojan. (July 2024). Retrieved July 13, 2024 from https://www.trendmicro.com/en_fi/research/24/d/trend-micro-collaborated-with-interpol-in-cracking-down-grandore.html.

[78] Trend Micro. 2022. Understanding the Patchwork Cyberespionage Group. (Mar. 2022). Retrieved March 6, 2022 from https://documents.trendmicro.com/assets/tech-brief-untangling-the-patchwork-cyberespionage-group.pdf.

[79] Jiang Ming, Dongpeng Xu, Yufei Jiang, and Dinghao Wu. [n. d.] BinSim: Trace-based Semantic Binary Diffing via System Call Sliced Segment Equivalence Checking. In.

[80] Abbas Naderi-Afooshteh, Yonghwi Kwon, Anh Nguyen-Tuong, Ali Razmjoo-Qalaei, Mohammad-Reza Zamiri-Gourabi, and Jack W Davidson. 2011. MalMax: Multi-Aspect Execution for Automated Dynamic Web Server Malware Analysis. In *Proc. 26th ACM CCS.* (Nov. 2011).

[81] Antonio Nappa, Zhaoyan Xu, M Zubair Rafique, Juan Caballero, and Guofei Gu. 2014. Cyberprobe: towards internet-scale active detection of malicious servers. In *Proc. 2014 NDSS.* (Feb. 2014).

[82] Terry Nelms, Roberto Perdisci, and Mustaque Ahamad. 2012. ExecScent: mining for new C&C domains in live networks with adaptive control protocol templates. In *Proc. 21st USENIX Security.* (Aug. 2012).

[83] Netresec. 2023. Publicly Available PCAP Files. (2023). Retrieved March 12, 2023 from https://www.netresec.com/?page=PcapFiles.

[84] James Newsome and Dawn Xiaodong Song. 2005. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. In *Proc. 12th NDSS.* (Feb. 2005).

[85] Carly Page. 2022. Rsocks, A popular proxy service, was just seized by the DOJ. (June 2022). Retrieved July 18, 2022 from https://techcrunch.com/2022/06/17/rsocks-proxy-seized-justice-department/.

[86] Nick Pantic and Mohammad Iftekhar Husain. 2015. Covert Botnet Command and Control Using Twitter. In *Proc. 31st Annual Computer Security Applications Conference (ACSAC).* (Dec. 2015).

[87] Fei Peng, Zhui Deng, Xiangyu Zhang, Dongyan Xu, Zhiqiang Lin, and Zhendong Su. 2014. X-force: Force-executing Binary Programs for Security Applications. In *Proc. 23rd USENIX Security.* (Aug. 2014).

[88] Jannik Pewny, Behrad Garmany, Robert Gawlik, Christian Rossow, and Thorsten Holz. 2015. Cross-Architecture Bug Search in Binary Executables. In *Proc. 36th IEEE Security and Privacy.* (May 2015).

[89] Stijn Pletinckx, Cyril Trap, and Christian Doerr. 2018. Malware Coordination Using the Blockchain: An Analysis of the Cerber Ransomware. In *Proceedings of the 6th IEEE Conference on Communications and Network security (CNS).* (May 2018).

[90] Checkpoint Research. 2019. Pony's C&C Servers Hidden Inside the Bitcoin Blockchain. (Dec. 2019). Retrieved March 6, 2022 from https://research.checkpoint.com/2019/ponys-cc-servers-hidden-inside-the-bitcoin-blockchain/.

[91] ESET Research. 2019. Casbaneiro: Dangerous Cooking with a Secret Ingredient. (Oct. 2019). Retrieved March 06, 2024 from https://www.welivesecurity.com/2019/10/03/casbaneiro-trojan-dangerous-cooking/.

[92] Edward J Schwartz, Thanassis Avgerinos, and David Brumley. 2010. All You Ever Wanted to Know About Dynamic Taint Analysis and Forward Symbolic Execution (But Might Have Been Afraid to Ask). In *Proc. 31th IEEE Security and Privacy.* (May 2010), 317–331.

[93] Securelist. 2022. The Dropping Elephant - Aggressive Cyber-Espionage in the Asian Region. (Mar. 2022). Retrieved March 6, 2022 from https://securelist.com/the-dropping-elephant-actor/75328/.

[94] David Shamah. 2014. How malware writers' laziness is helping one startup predict attacks before they even happen. (Oct. 2014). Retrieved March 16, 2024 from https://www.zdnet.com/article/how-malware-writers-laziness-is-helping-one-startup-predict-attacks-before-they-even-happen/.

[95] Paria Shirani, Lingyu Wang, and Mourad Debbabi. 2017. Binshape: Scalable and Robust Binary Library Function Identification using Function Shape. In *Proceedings of the 14th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA).* (July 2017).

[96] Standford.Edu. 2022. A System for Detecting Software Similarity. (Nov. 2022). Retrieved Feburary 26, 2024 from https://theory.stanford.edu/~aiken/moss/.

[97] Tsuyoshi Taniguchi, Harm Griffioen, and Christian Doerr. 2021. Analysis and Takeover of the Bitcoin-Coordinated Pony Malware. In *Proc. 16th ACM Symposium on Information, Computer and Communications Security (ASIACCS).* (June 2021).

[98] ASERT Team. 2018. Donot Team Leverages New Framework. (Mar. 2018). Retrieved March 09, 2023 from https://www.netscout.com/blog/asert/donot-team-leverages-new-modular-malware-framework-south-asia.

[99] Counter Threat Unit Research Team. 2017. Bronze Butler Targets Japanese Enterprises. (Oct. 2017). Retrieved March 06, 2024 from https://www.secureworks.com/research/bronze-butler-targets-japanese-businesses.

[100] Florian Tegeler, Xiaoming Fu, Giovanni Vigna, and Christopher Kruegel. 2012. Botfinder: finding bots in network traffic without deep packet inspection. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies.* (Oct. 2012).

[101] Nikola Todorovic and Abhi Chaudhuri. 2023. Using AI to help organizations detect and report child sexual abuse material online. (2023). Retrieved March 12, 2023 from https://blog.google/around-the-globe/google-europe/using-ai-help-organizations-detect-and-report-child-sexual-abuse-material-online/.

[102] Milad Torkashvan and Hassan Haghighi. 2015. CB2C: A Cloud-Based Botnet Command and Control. *Indian Journal of Science and Technology.*

[103] Vitor Ventura. 2019. Gustuff Banking Botnet Targets Australia. (Apr. 2019). Retrieved March 06, 2024 from https://blog.talosintelligence.com/2019/04/gustuff-targets-australia.html.

[104] VirusTotal. 2004. VirusTotal. (June 2004). Retrieved January 05, 2024 from https://www.virustotal.com/.

[105] Ryan Vogt, John Aycock, and Michael J Jacobson Jr. 2007. Army of botnets. In *Proc. 14th NDSS.* (Feb. 2007).

[106] Shuai Wang and Dinghao Wu. 2017. In-memory Fuzzing for Binary Code Similarity Analysis. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE).* (Oct. 2017).

[107] WordPress. 2024. WordPress.com: Build a Site, Sell Your Stuff, Start a Blog & More. (2024). Retrieved December 13, 2024 from https://wordpress.com/.

[108] Dongpeng Xu, Jiang Ming, and Dinghao Wu. 2017. Cryptographic Function Detection in Obfuscated Binaries via Bit-Precise Symbolic Loop Mapping. In *Proc. 38th IEEE Security and Privacy.* (May 2017).

[109] Dongpeng Xu, Jiang Ming, and Dinghao Wu. 2017. Cryptographic function detection in obfuscated binaries via bit-precise symbolic loop mapping. In *Proc. 38th IEEE Security and Privacy.* (May 2017).

[110] Haichuan Xu, Mingxuan Yao, Runze Zhang, Mohamed Moustafa Dawoud, Jeman Park, and Brendan Saltaformaggio. 2024. Dva: extracting victims and abuse vectors from android accessibility malware. In *Proc. 33rd USENIX Security.* (Aug. 2024).

[111] Zhaoyan Xu, Antonio Nappa, Robert Baykov, Guangliang Yang, Juan Caballero, and Guofei Gu. 2014. Autoprobe: Towards Automatic Active Malicious Server Probing Using Dynamic Binary Analysis. In *Proc. 21st ACM CCS.* (Nov. 2014).

[112] Mingxuan Yao, Jonathan Fuller, Rajita Pai Sridhar, Saumya Agarwal, Amit K. Sikder, and Brendan Saltaformaggio. 2023. Hiding in Plain Sight: An Empirical Study of Web Application Abuse in Malware. In *Proc. 32nd USENIX Security.* (Aug. 2023).

[113] Mingxuan Yao, Runze Zhang, Haichuan Xu, Ryan Chou, Varun Chowdhary Paturi, Amit Kumar Sikder, and Brendan Saltaformaggio. 2024. Pulling Off The Mask: Forensic Analysis of the Deceptive Creator Wallets Behind Smart Contract Fraud. In *Proc. 45th IEEE Security and Privacy.* (May 2024).

[114] Runze Zhang, Ranjita Pai Sridhar, Mingxuan Yao, Zheng Yang, David Oygenblik, Haichuan Xu, Vacha Dave, Cormac Herley, Paul England, and Brendan Saltaformaggio. 2025. Identifying incoherent search sessions: search click fraud remediation under real-world constraints. In *Proc. 46th IEEE Security and Privacy.* (May 2025).

[115] Runze Zhang, Mingxuan Yao, Haichuan Xu, Omar Alrawi, Jeman Park, and Brendan Saltaformaggio. 2025. Hitchhiking vaccine: enhancing botnet remediation with remote code deployment reuse. In Proceedings of the Network and Distributed System Security (NDSS) Symposium.

[116] Shuang Zhao, Patrick PC Lee, John CS Lui, Xiaohong Guan, Xiaobo Ma, and Jing Tao. 2012. Cloud-Based Push-Styled Mobile Botnets: A Case Study of Exploiting the Cloud to Device Messaging Service. In *Proc. 28th Annual Computer Security Applications Conference (ACSAC).*

[117] Shuofei Zhu, Jianjun Shi, Limin Yang, Boqin Qin, Ziyi Zhang, Linhai Song, and Gang Wang. 2020. Measuring and Modeling the Label Dynamics of Online Anti-Malware Engines. In *Proc. 29th USENIX Security.* (Aug. 2020).

A Anti-Analysis APIs

Malware uses anti-analysis and defensive evasion APIs to avoid detection. VADER hooks these APIs to facilitate malware exploration. The APIs used in VADER's design are listed in VADER's code repository [58]. These APIs were identified through manual malware analysis, industry reports [7, 34], and prior research on system emulation and evasion techniques [55, 71, 38].

B De-manipulation Algorithm Source Code Similarity

We use Moss [96] to confirm that each decoding algorithm in VADER is distinct. The comparison results are available in VADER's code repository [56]. As shown in Row 6, no algorithms overlap, where $|R|$ is the set of all de-manipulation algorithms, and m, n index different algorithms. Matches occurred only when an algorithm was compared with itself. This confirms that each de-manipulation algorithm in VADER is uniquely implemented.

C De-Manipulation Comparison Graphs

We made the symbolic expression of de-manipulation algorithm comparisons available in VADER's code repository [57]. For example, Column 1, Row 1 shows comparisons for three Base16 versions, totaling nine comparisons (e.g., Base16v1 vs. Base16v2, etc.). Each graph covers a two-hour runtime (X-axis), with lines representing de-manipulation similarity (Y-axis). A black vertical line marks the latest plateau, with all comparisons plateauing for over 30 minutes. Plateaus occurred between 20 and 85 minutes. We selected the best algorithm from each class for VADER to identify malware de-manipulation algorithms.

D Web App Accounts Identified In Our Study

The dead drops identified by VADER in this research have been made publicly available in VADER's code repository [59].