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

Mingxuan Yao, *Georgia Institute of Technology;* Jonathan Fuller, *United States Military Academy;* Ranjita Pai Kasturi, Saumya Agarwal, Amit Kumar Sikder, and Brendan Saltaformaggio, *Georgia Institute of Technology*

https://www.usenix.org/conference/usenixsecurity23/presentation/yao-mingxuan

This paper is included in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

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

Mingxuan Yao[1], Jonathan Fuller[2], Ranjita Pai Sridhar[1], Saumya Agarwal[1], Amit K. Sikder[1], Brendan Saltaformaggio[1]
[1]*Georgia Institute of Technology* [2]*United States Military Academy*

Abstract

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

1 Introduction

Web applications provide a wide range of utilities, including content delivery [1]–[5], data storage [5]–[8], social networking [9]–[11], and much more. These utilities incentivize malware authors to integrate popular web apps [12]–[15] into their malware. This integration enables Web App-Engaged (WAE) malware to profile a victim's system, retrieve additional payloads, exfiltrate and store victim information, and even hide subsequent connections to attacker-controlled C&C servers. Even worse, WAE malware are stealthier because their network traffic blends in with non-malicious traffic. Stopping this abuse requires technical and policy solutions from Incident Responders (IRs) and web app providers. However, such novel collaboration has not happened to date, and little research has been done to prove that WAE malware are prevalent enough to warrant such an investment.

Recent works on web app security show the possibility of abusing benign services for malicious purposes [16]–[20]. Yet, these works only scratch the surface of the WAE malware problem by solely considering proof-of-concepts [16]–[18] of possible abuse. The research community also proposed feature sets based on user engagement [19]–[24] to classify abuse, where *engagement* is the interaction (e.g., views, likes, commits) that end-users have with web app content. Unfortunately, these ad-hoc studies focused on specific web apps and omitted the WAE malware required to attribute the abuse. Separately, prior research performed case studies of individual WAE malware samples [25]–[27] but only provided anecdotal evidence.

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

RQ1: How prevalent are WAE malware?
RQ2: What capabilities do web apps provide attackers?
RQ3: How effective is the current collaboration between IRs and web app providers?
RQ4: Can the abused web app help estimate the extent of a WAE malware's infection?
RQ5: How should IRs and web app providers collaborate to effectively remediate WAE malware?

To answer our research questions, we studied WAE malware in collaboration with Netskope, the leading Secure Access Service Edge (SASE) provider, which provides cloud security and networking to more than 25% of the Fortune 100. Together we built Marsea,[1] an automated concolic analysis pipeline, to perform a scalable and retroactive study of WAE malware and answer our five **RQs**. Given only a malware binary, Marsea uses a novel *session reconstruction* and *web app*

[1]**M**alw**A**re **R**eplace malicious **S**ervers with w**E**b app **A**buse.

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

To evaluate the prevalence of WAE malware, we deployed Marsea on 3K samples pulled randomly from VirusTotal [28] spanning the last 3 years. From this dataset, Marsea found that the number of web app abuse increased by 226% since 2020. Further, a more comprehensive study requires a larger set of known WAE malware. To this end, our collaborator gave us an additional 7K suspected WAE malware samples to augment our existing dataset. Marsea's study of all 10K samples uncovered 893 malware in 97 malware families abusing 29 web apps. Worse still, Marsea revealed significant delays in the current collaboration between IRs and web app providers. We found an average delay of 253 days between when WAE malware authors create their assets on the web apps and when the WAE malware is finally detected. Our findings also suggest that WAE malware could have infected up to 909,788 victims by evaluating the engagement data from 33 abused web app assets. At the time of writing, we have used Marsea to collaborate with the web app providers to take down 50% of active WAE assets, and we are awaiting responses for the remainder. Lastly, we have made Marsea available at: `https://github.com/CyFI-Lab-Public/MARSEA`.

2 Challenges and Motivation

RQ1: A temporal measurement of malware web app adoption is required to answer **RQ1**. However, many web app assets used by malware are no longer available. Thus, failing to build this connection leaves subsequent network communications undiscoverable. Worse still, tracking only web app connections is not evidence of abuse (e.g., malware often connect to web apps to test network connectivity). Motivated by this, Marsea performs concolic analysis with a customized exploration strategy to spoof network communications. During our preliminary study, Marsea concolically analyzed the Zusy WAE malware and confirmed its connection to the Pastebin web app [5]. Then, Marsea identifies that the fetched content is written to a local file, confirming that Zusy is not using Pastebin for a connectivity check. In fact, Zusy hosts the configuration of the cryptocurrency miner XMRIG in the Pastebin post. Further, Marsea confirmed that Zusy does not rely on attacker-controlled C&C servers and only abuses Pastebin during its execution.

RQ2: Armed with Marsea, our study (§5) revealed that WAE malware are increasing in prevalence, which should alert the IRs and the web app providers to be prepared for future growth. Ideally, proof-of-abuse, including the web app capabilities being abused by each identity and asset, is needed to enable rapid remediation. Deriving this proof motivated our measurement of capabilities that web apps provide to malware **RQ2**.

Capability identification requires partitioning the WAE malware's abuse vectors based on their identity and assets. To identify the abuse vectors, Marsea uses *Vector Rules* (§3.2) to match the information flow from the fetched content. By tracking the information flow from the content fetched by Zusy to the `WriteFile` operation, Marsea identified the File Download vector. Unfortunately, partitioning is challenging since the identity and asset are not readily available in the WAE malware binary. Investigating Pastebin abuse requires the Post's content (the Pastebin *asset*) and the malware author's Username (the Pastebin *identity*). Upon investigation, we found that Zusy's binary includes no asset or identity but only a Pastebin paste key as the object name when configuring the network session with Pastebin. To overcome this challenge and obtain the identity and asset used by Zusy, Marsea performs session reconstruction (§3.3) to reproduce the Pastebin session (i.e., `GET pastebin.com/raw/QKAdpkY`). Then, Marsea uses Web App Modeling (§3.4) to parse the session and extract the paste key `QKAdpkY`. Given the identified key and modeling, Marsea requests Pastebin and confirms that Zusy is abusing the file hosting capability of Pastebin via the asset (a post named `active`) from the identity `anthrax_`.

RQ3: Next, we turned our attention to studying how effective the current collaboration between IRs and web app providers is. To do so, we define two key time windows during a WAE malware outbreak. The *Response Delay* (*RD*) is the time between when the attackers create the assets to when the corresponding WAE malware (that abuse those assets) are first submitted to VirusTotal [28]. The *Post-Detection Window* (*PW*) begins when the WAE malware is first submitted to VirusTotal and ends when web app providers take down the associated assets. Minimizing *RD* and *PW* indicates a more effective collaboration. Upon closer investigation, we found that web apps track the time information of the hosted assets and often expose such data to the public. Motivated by this, Marsea queries the time information of assets from the web app and calculates the *RD* of Zusy as 21 days and *PW* as 108 days. Worse still, when we began this research, the post was still available on Pastebin. During our study, Marsea revealed a high *RD* and *PW*, leading to significantly more infected victims.

RQ4: Given that the current collaboration fails to quickly take down abused assets, we wondered how many victims end up infected by WAE malware. Several works have proposed covert access [29] or active monitoring [30], [31] of attacker-controlled servers. However, IRs still need to collect information manually to evaluate the extent of infections. This is hardly scalable and not applicable to WAE malware since the malware's activity takes place on a public web app. In fact, the publicly available data gives IRs and web app providers a unique opportunity to monitor the extent of a WAE malware family's infection by harvesting the web app's engagement data (e.g., views, comments). Armed with this insight, Marsea harvested the view numbers of the Pastebin post abused by Zusy malware, which indicates up to 796 victims.

RQ5: Noting the deficiency of current collaboration efforts (**RQ3**) and the extent of WAE malware infections (**RQ4**), we develop our techniques in Marsea to target rapid remediation of WAE malware. For Zusy, we submitted proof-of-abuse derived from Marsea to Pastebin, resulting in the revocation of abused identities and assets. Apart from Zusy, Marsea enabled us to correlate the assets attributed to WAE malware with other identities and assets on the web app to facilitate the remediation (designated *Lateral Remediation* in §5.5). Moreover, modern malware perform their infection via multiple stages, and WAE malware often fetch these stages from web apps. In fact, this research has found that this infection chain can reveal the connections among different WAE malware families and can assist IRs in uncovering new WAE malware from a single WAE sample (designated *Early Stage Termination* in §5.5). Lastly, Marsea also exposes an opportunity for us to monitor and counteract the migration of WAE malware intra- and inter-web apps (designated *Migration Remediation* in §5.5).

3 Methodology

Marsea needs only a malware sample binary as input. The analysis is automated to identify WAE malware and output proof-of-abuse and answers the 5 **RQs**. Marsea is open source and available at: <redacted>.

3.1 Concolic Exploration Strategy

Marsea uses concolic analysis to spoof network sessions to ensure forced execution of the malware. Specifically, Marsea introduces symbolic data through system *input* APIs that read data from external sources (e.g., network communication). The symbolic data enables Marsea to drive the execution when the assets are unavailable. As stated in §2, the Zusy malware abused Pastebin to host a malicious configuration file. During analysis, Zusy invokes `InternetReadFile` to fetch the content from Pastebin. Marsea introduces symbolic data to the buffer defined by `InternetReadFile`. When Zusy checks the string (e.g., `cpu`) in the buffer, Marsea can generate two states to ensure continued execution even if the abused asset is unavailable.

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

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

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

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

3.2 Web App and Vectors Identification

Concolic analysis proves efficient for selective path exploration, but, on its own, it cannot answer our 5 **RQs**. To answer **RQ1**, Marsea needs to identify abused web apps and the abuse vectors performed by the malware. This is similar to a detective (e.g., IRs)

Algorithm 1: Marsea's Vector Identification.

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

collecting evidence from a crime scene (e.g., malware) for off-site processing. Without the context of the crime scene, the detective may make (inaccurate) assumptions about evidence. To overcome this challenge, Marsea uses a fined-grained and context-driven approach to identify abused web apps and abuse vectors.

Algorithm 1 shows how Marsea identifies abused web apps and vectors. During concolic analysis, Line 3 of Algorithm 1 tracks the invocation of system APIs which build the connection to remote endpoints (e.g., `WinHttpConnect`), denoted as Π. Once the malware invokes a system API in Π, Marsea recovers the communication target information from the input to the API ($GetTarget()$ of Algorithm 1). For example, when Zusy builds the network connection to the remote endpoint by invoking `InternetConnect`, Marsea identifies the endpoint `pastebin.com` from parameter `lpszServerName`. Notably, since Marsea's concolic analysis logic needs no information of Π annotation, it is a one-time effort and can be extended easily. Armed with the ability to identify abused web apps, Marsea can pursue large-scale analysis to identify the prevalence of WAE malware (**RQ1**).

Definitions for Abuse Vector Rules	
Primitive	**Example Library Functions**
Receive:	`WinHttpReadData` | `WinHttpReceiveReponse` | ...
ReadFile:	`ReadFile` | `ReadFileEx` | `ReadFileScatter` | ...
SysOp:	`RegOpenKey` | `CreateFile` | `DeleteFile` | ...
WriteFile:	`WriteFile` | `fwrite` | `write` | ...
Send:	`send` | `HttpSendRequest` | `WinHttpSendRequest` | ...
Connect:	`WinHttpConnect` | `InternetConnect` | `connect` | ...
Execute:	`ShellExecute` | `CreateProcess` | ...

Abuse Vectors	Priority	Rules
Execute	R_{11}	$(Receive \rightarrow WriteFile) \rightarrow Execute$
File Download	R_{12}	$Receive \rightarrow WriteFile$
Message Fetch	R_{13}	$Receive \rightarrow SysOp$
File Upload	R_{21}	$ReadFile \rightarrow Send$
Message Push	R_{22}	$SysOp \rightarrow Send$
Jump Server	R_{31}	$Connect \rightarrow Receive \rightarrow Connect'$

Table 1: Abuse Vectors And Associated Rules.

Answering **RQ2** requires Marsea to identify the abuse vectors, which indicates the capabilities the web apps provide. To do so, Marsea tracks information flow in the context of malware execution. Unfortunately, tracking all concrete data during analysis is unnecessary and unscalable. Recall the symbolic data Marsea introduced to enable the concolic analysis (§3.1). It turns out that the propagation of symbolic data indicates the information flow. As shown in Lines 10 to 13 of Algorithm 1, Marsea tracks symbolic data propagation, saved in *Prop*. For example, after Marsea introduces symbolic data through `InternetReadFile` invoked by Zusy, the symbolic data is propagated to `WriteFile`. In this case, Zusy is abusing the web app to perform the *File Download* vector. These API dependencies Marsea uses (e.g., `InternetReadFile` $\rightarrow$ `WriteFile`) are presented as vector rules, R_{ij} in Table 1. The full list of system APIs will be open-sourced along with Marsea. We present the definition of each abuse vector below:

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

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

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

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

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

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

As shown in Table 1, we define the rule primitives based on the vectors, which represent a set of APIs that perform a task that the primitive is named for. Column 1 (bottom half of the table) lists the abuse vectors considered in this study. To decide on the abuse vector rules, we referenced reports from security experts [32], [33] and the MITRE ATT&CK Framework [34] to understand how malware vectors are generally applied. Column 2 presents the matching priority for each vector. Specifically, $R_{i,j}$ has a higher priority than $R_{i,m}$ when $j < m$. Given *Prop* and the Rule set, Marsea identifies abuse vectors by evaluating if the path presented by R_{ij} exists in *Prop* (Line 21 of Algorithm 1). Since the exploration logic does not depend on vector identification, vector rules are both *composable* and *extensible* with no need to re-analyze.

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

3.3 Session Reconstruction

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

However, given the multi-path nature of concolic analysis, acquiring in-order network sessions at the operating system layer is challenging. For example, concolic analysis can switch the exploration state after Zusy invokes `InternetConnect`. Then, on the system level, the new network sessions appear to be appended after `InternetConnect` of the old state, which is not true in the malware. To overcome this challenge, we introduce a graph automata model of network session states, which enables Marsea to reconstruct sessions by interleaving different exploration states. As shown in Figure 1, we generalized the session states into opened ($\textcircled{o}$), connected ($\textcircled{c}$), sent ($\textcircled{s}$), and closed ($\textcircled{cl}$). The transitions between different states fall into five categories: (1) a transition from the opened state to a connected state by connecting to the target (function $h_o.c()$), (2) a transition from the old connected state to a new connected state by assigning session parameter information through headers (function $h_c.a()$) and parameters (function $h_c.p()$), (3) a transition from the connected state to sent state by sending the request (function $h_c.s()$), (4) a transition to the closed state by freeing the resource (function $h_s.cl()$ and $h_o.cl()$). Each transition function in Figure 1 presents a group of system APIs. During the analysis of Zusy, the session state transition $\textcircled{o} \rightarrow \textcircled{c}$ happens when Zusy invokes `InternetConnect`. As stated in §3.2, Marsea extracts the abused web app, e.g., `pastebin.com`, during this transition. The next session state transition Marsea observed in Zusy was $\textcircled{c}$ to $\textcircled{c}$, caused by invoking `InternetOpenRequest`. During this transition, Marsea extracts the session parameters. For Zusy, this is the verb (`GET`) and object name (`pQKAdpkY`). When Zusy invokes `HttpSendRequest`, the session state transitions from $\textcircled{c}$ to $\textcircled{s}$. Note the transition from $\textcircled{s}$ to $\textcircled{cl}$ can include more operations (e.g., receiving). We abstracted those into a dotted line since Marsea only needs sessions composed by the malware to extract the abused identities and assets. Given Marsea's session graph automata model, it reconstructs multiple network sessions by interleaving execution states.

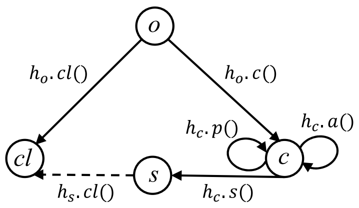


Figure 1: Graph Automata Model To Guide Session Reconstruction From Interleaving Execution States.

3.4 Web App Modeling

Given reconstructed network sessions, Marsea is ready to identify the identities and assets. However, since web apps do not share the same session syntax, it is challenging to parse the reconstructed network sessions. Luckily, each web app provides public documents to regulate the network sessions. Motivated by this, we proposed a web app model, $\Phi = (\phi_\alpha, \phi_\beta)$, to model the API described in the web app's documentation. ϕ_α includes the compositions of the public frontends of the web apps. $\phi_\beta = [\theta_1, \theta_2, ...]$ includes web app APIs. Each web app API θ_i embeds the response format to annotate the identity, asset, time, engagement, and raw data. Now, Marsea can uncover the web app's identifier specific to the abused asset from the reconstructed network sessions. Then, Marsea uses the identifier and web app APIs, ϕ_β, to harvest and parse the information fetched from the web app. For example, to query the abused post from Pastebin through the public frontend, Zusy appends a Pastebin paste key `QKAdpkY` after the URL `pastebin.com/raw`. Using ϕ_α, Marsea extracts the paste key from the reconstructed network sessions. Meanwhile, Pastebin APIs `api_scrape_item` and `api_scrape_item_meta` can be used to query the raw data and the metadata information, respectively. By modeling these APIs, Marsea queries Pastebin using the paste key and parses the response to uncover: (1) the raw data of the abused post, (2) the identity (Username `anthrax_`), (3) asset (post named `active`), (4) creation time (12-19-2021), and (5) the engagement data (796 views).

With the identity, Marsea uses the modeled frontend (ϕ_α) and modeled APIs (ϕ_β) to query the frontend and APIs accepting the identity as input. For Zusy, Marsea uses modeled frontend `pastebin.com/u/` to query other posts from `anthrax_`. Notably, given the finite number of web apps and the APIs they provided, Marsea can be easily extended to support additional web apps. Till now, the partitioned vectors constitute the admissible proof-of-abuse, which enables fast collaboration. (**RQ5**). Meanwhile, Marsea can evaluate the extent of infection using the harvested engagement data (**RQ4**). By querying the first detection date of the malware and comparing the date with the harvested time information, Marsea can calculate the Response Delay and the Post-detection Window (**RQ3**).

4 Validating Our Techniques

Implementation. Marsea is implemented in C++ (12K lines) leveraging S2E [35], which has been used in top-tier research, for concolic analysis [36]–[38], and Python (8K lines) to attribute identified malicious vectors to abused identities and assets and harvest corresponding engagement information on the abused web app platforms. We used AVClass2 [39], a state-of-the-art malware labeling tool, to label our dataset. We used an Ubuntu 20.04 LTS system to host Marsea. Given one malware, Marsea spawns a Windows 7 virtual machine (20GB RAM and 6 x 3.9GHz CPUs) based on the snapshot of a bare-metal machine to mimic the victim system.

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

4.1 Vector Identification

Table 2 presents Marsea's abuse vector identification evaluation. Columns 1-4 list the abused web apps, malicious vectors, malware families, and the number of samples (#V) in our ground truth dataset. Column 5 presents the ground truth results (G) derived from the industry report and manual investigation. Column 6 presents the concrete execution results (C). Specifically, we count the vectors the concrete execution *could* reveal by manually verifying the information flow. The results generated by Marsea (M) and the evaluation metrics are presented in Columns 7-10, including True Positive (TP), False Positive (FP), and False Negative (FN). The paths Marsea explored are shown in the last column.

Overall, Marsea correctly (TP) identified 40 abuse vectors across 20 malware samples. As shown in Column 5 of Table 2, compared with single-path concrete execution, Marsea identified 37 more vectors, accounting for 86% of the vectors in the ground truth.

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

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

1,2,3: Short for Stellarstealer, Darkkomet and VirusTotal
4: Average explored paths for each sample
#ID: Number of identities   #A: Number of assets

Table 2: Evaluation Of Abuse Vector Identification And Identity and Assets Attribution.

Understandably, since remote endpoint network communication can be implemented through multiple API calls (e.g., `socket`, `connect`, `recv`), single-path exploration can halt during connection establishment if the endpoint is unavailable. Next, our manual investigation found 43 abuse vectors, revealing an overall accuracy of 93%. As seen in Row 2, Row 5, and Row 17 of Table 2, Marsea produced 2 FPs and 3 FNs when it analyzed Apost and Msil. For each variant of Apost, our manual investigation revealed that the malware retrieved 3 files from a GitHub repository. One of these files is a binary executed by the malware, which accounts for the 1 Execute (EXE) abuse vector, while the other 2 files were dropped as configuration files. However, Marsea reported no EXE vector (Row 2) and 3 File Download (FD) vectors (Row 5). Similarly, for one Msil variant, our manual investigation uncovered Pastebin abuse to resolve the new network connection target. However, Marsea reported no Jump Server (JS) vector. Upon closer inspection, we found that both cases were due to unresolved symbolic constraints accumulated through exploration, which further impedes Marsea's exploration. We confirmed these are rare occurrences. Note, since the FD vector is the precondition of the EXE vector (the malware needs to download the file before executing it), in the case of Apost, the FP is the direct cause of the FN. Therefore, once Marsea can escalate FD vector to EXE vector, it will not introduce any FPs or FNs. Overall, Marsea was 93% accurate in abuse vector identification, making it robust for our large-scale study.

4.2 Identities & Assets Attribution

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

Next, Marsea introduced 3 FNs when analyzing malware Apost and Stellarstealer. The FN in Apost is caused by the prior vector identification FN in §4.1 (if Marsea misses a vector, then it can not attribute that vector). Upon closer inspection, we found that the FNs in Stellarstealer are because the abused Telegram channel is no longer available. Even though Marsea can extract the asset's content (Telegram channel) from the network session, Marsea cannot identify the identity (i.e., Username) since that information is hosted in the channel and not included in the network sessions. Still, given the low number of FNs and over 93% accuracy, Marsea provides the means to attribute abuse vectors to identities and assets effectively.

5 WAE Malware Study Insights

To study the prevalence of WAE malware (**RQ1**), we deployed Marsea on 3K malware randomly pulled from VirusTotal [28], spanning 2020-2022 (1K samples per year). Specifically, we used VirusTotal Intelligence Search [40] to fetch signatures of all Windows PE malware with a first submission date in each of the 3 years and at least 5 AV engine detections, indicating maliciousness [41]. Then, we randomly selected 1K signatures from each year and deployed Marsea on the fetched samples. For **RQs 2-5**, we extend our dataset with an additional 7K suspected WAE malware samples collected by Netskope (totaling 10K samples) to investigate WAE malware-specific topics.

5.1 WAE Malware Prevalence

Marsea extracted communication endpoints from the 3K samples randomly pulled from VirusTotal. Based on

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

Table 3: RQ1 Malware Reliance On WAE And Year Over Year (YoY) Increase Of WAE Adoption.

the endpoints, the malware are categorized as WAE-only (i.e., communicates solely with web apps), hybrid (i.e., communicates with both web apps and attacker-controlled C&C servers), and malicious backend-only (i.e., communicates solely with attacker-controlled C&C servers). Marsea evaluates the reputation of the endpoint effective Second-Level Domain (eSLD) using Tranco 1M top sites [42] and Cisco Talos [43]. If the eSLD is categorized as *untrusted* by Cisco Talos or is not listed in the Tranco top sites, Marsea considers it an attacker-controlled C&C server.

Notably, malware can contact endpoints only to verify the network connectivity of the infected victim system. Failing to exclude these cases would lead to a biased prevalence measurement. However, Marsea would not report any abuse vector for a connectivity check since the information flow (e.g., status code) fetched from the websites would not flow to the system APIs Marsea modeled. The overall results are presented in Table 3. We made several observations from Table 3: (1) As shown in the Total row of Table 3, in the last three years, malicious backend-only malware, WAE-only malware, and hybrid malware account for 2598 (86.6%), 367 (12.2%), and 35 (1.2%) of all 3K malware. (2) As shown in Column 5 of Table 3, the malware using only malicious backends has decreased by 13.7%. (3) Marsea found notably few hybrid malware (only 35 across all 3 years), which suggests that WAE malware authors prefer to shift entirely to web apps. (4) From the Total row of Table 3, the WAE-only malware has increased by 226% in the last three years (increased by 22% and 166% in 2022 and 2021), indicating the recent prevalence of WAE-only malware.

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

Category	Web Apps	Vector	Mal	Fam	*FAD*	*LAD*	History (Days)
File Sharing	GDrive	EXE	263	24	2020-01	2022-06	894
		FD	36	6	2020-03	2022-03	716
		JS	23	5	2019-06	2022-03	1,002
	Dropbox	EXE	13	6	2010-07	2020-10	3,734
		FD	1	1	2016-05	2022-02	2,091
		FU	4	3	2016-09	2021-07	1,756
	Pastebin	FD	5	3	2021-12	2022-04	130
		MF	26	11	2018-02	2022-02	1,463
		JS	25	10	2014-02	2022-04	2,990
	Zippyshare	EXE	1	1	2018-06	2018-06	3
	Imgur	EXE	1	1	2013-10	2013-10	1
	Tietuku	EXE	17	1	2016-05	2020-06	1,490
		FD	17	1	2016-05	2020-06	1,490
Software Hosting	GitHub	EXE	29	10	2020-09	2022-10	772
		FD	4	2	2020-09	2022-10	762
		MF	9	5	2019-10	2022-04	922
		JS	4	1	2020-03	2022-03	732
	SourceForge	EXE	9	1	2022-01	2022-02	33
		JS	11	6	2022-01	2022-02	20
Social Networking	Telegram	MF	4	2	2022-02	2022-03	30
	Twitter	JS	54	7	2014-06	2022-01	2,773
	Discord	EXE	49	11	2021-03	2022-01	314
		MP	37	9	2021-10	2022-03	153
	Slack	MP	13	4	2020-08	2022-03	585
	Facebook	MF	1	1	2018-07	2018-08	29
	V Kontakte	FD	1	1	2018-02	2020-01	705
Website Builder	Wordpress	EXE	3	2	2011-11	2014-05	914
		JS	9	3	2011-05	2020-05	4,026
	Webs	EXE	5	2	2010-03	2019-10	3,596
		FD	3	1	2012-05	2017-11	2,021
	Tripod	EXE	1	1	2007-03	2014-05	2,628
		FD	1	1	2012-01	2021-07	3,474
Blog	Blogspot	EXE	1	1	2018-07	2018-08	853
		FD	1	1	2011-11	2014-05	932
		MF	1	1	2018-07	2018-08	29
		JS	3	3	2017-05	2020-02	1,014
Dynamic DNS	Afraid	JS	110	8	2020-09	2022-03	544
	DuckDNS	JS	8	1	2022-02	2022-02	1
Local Info	Cmyip	MF	6	2	2011-11	2014-05	910
	Wikidates	MF	5	1	2022-03	2022-03	1
	Ip-api	MF	3	2	2020-10	2020-12	59
	wimip	MF	2	2	2011-11	2014-05	898
	Ip2location	MF	2	2	2011-11	2014-05	898
	Ip-address	MF	2	2	2011-11	2014-05	898
	Ipify	MF	11	6	2022-01	2022-02	27
	Timeanddate	MF	1	1	2022-03	2022-03	1
Anti-Virus	VirusTotal	FU	106	10	2014-06	2022-05	2,281
Summary	29	6	893 [1]	97[1]	2007-03	2022-03	5,493

Table 4: WAE Vector Landscape.

5.2 WAE Malware Capabilities

Deploying Marsea on 10K malware samples confirmed 893 WAE malware across 97 malware families, abusing 29 web apps. Since WAE malware abuse vectors indicate the web app capabilities abused by the malware, we present the vectors Marsea identified in Table 4. We aimed to study the temporal adoption of each abuse vector on different web apps. To do so, we computed the First Abuse Date (FAD) and Last Abuse Date (LAD) for each vector v on every web app w, shown in Columns 6-7. The FAD is the earliest VirusTotal "first submission date" across all WAE malware samples found abusing v on w. The LAD is the most recent VirusTotal "first submission date" across all WAE malware samples abusing v on w (i.e., when VirusTotal discovered the most recent variant). Then, we measured the time difference to evaluate the abuse history of malware abusing v on w, shown in the History column of Table 4. Notably, the lifetime of abused assets can be longer than the abuse history, which will be evaluated in §5.3.

As shown in the Summary row of Table 4, Marsea found a total of 893 (8.9%) WAE malware in 10K samples. Notably, 32% of the 7K samples that Netskope suspected as being WAE malware were only performing network connectivity checks using web apps. As mentioned in §5.1, connectivity checks present no malicious vectors and are omitted from our study. This accounts for the drop in the percentage of WAE malware compared with §5.1.

Row 1 Columns 4-5 of Table 4 show that 263 (29.4%) WAE malware in 24 (24.7%) families abuse GDrive to perform the Execute (EXE) vector. GDrive is the most abused web app, illustrating the attackers' preference for reputable web apps. Surprisingly, GDrive not only gives the WAE malware the capability of hosting malicious executables, as shown in Row 1, but also enables the resolution of next-hop endpoints, shown as JS vector in Row 3. Similarly, from Rows 37-38 (Dynamic DNS), Afraid and DuckDNS provide 118 WAE malware with the capability to resolve additional endpoints, protecting attacker-controlled C&C servers. Consequently, when the resolved endpoints are denylisted, the attackers can update the record in the web app to re-activate the WAE malware on the victim system.

From the category Local Info, Marsea found 8 web apps abused to query the IP, geo-location, and the local time of the victim system. Since identity and assets are not required to use these services, it is challenging to remediate the abuse. We suggest that these web app providers require an identity in the network session when providing service to prevent abuse by blocking the malicious identity. The wide range of capabilities and the benefit of the benign traffic incentivizes malware authors to integrate web apps, which explains the growth of WAE malware found in §5.1. Unnervingly, we found that this incentivization happened earlier than we thought. As shown in the Summary row of Table 4, the WAE malware can be traced back to 2007, and the average abuse history of web apps can reach up to 5,493 days.

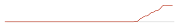
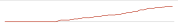
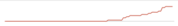
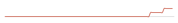
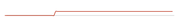
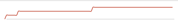
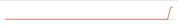
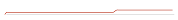
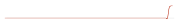
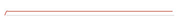
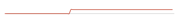
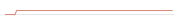
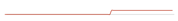
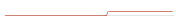
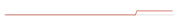
Web Apps	Live Mal	#ID	#A	Live Assets Growth (2011-2022)	Response Delay				μ_{pw} (Days)
					Min	Max	Med	Avg	
GDrive	214	41	42		2	104	72	59	-59
GitHub	34	19	27		0	1,320	446	583	173
Pastebin	33	12	13		1	242	122	118	-118
Telegram	2	2	2		296	359	327	327	-328
Twitter	26	1	1		-51	-51	-51	-51	51
WordPress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

Table 5: Collaboration Evaluation By Measuring RD And PW of Live Identity (ID) and Asset (A).

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

5.3 Collaboration Evaluation

To evaluate collaboration effectiveness, we measure RD by calculating the time difference between the First Creation Date (FCD) of assets the WAE malware abused and the First Submission Date (FSD) of the WAE malware on VirusTotal. RD also presents the time window that the WAE malware *could* have to infect systems without being identified as malware. Similarly, we calculate PW using the time difference between the malware's First Submission Date (FSD) and the attacker's Last Update Date (LUD) of the assets the malware abused. Notably, Marsea uses web app modeling (§3.4) to harvest the information. Since the web app abuse identification does not rely on web app modeling, we curated the web app modeling based on the identified web apps during our study. The web apps and their available information are listed in Appendix A.

The results of this evaluation are shown in Table 5. Note that WAE malware can abuse legitimate web app assets as its components (e.g., pulling legitimate proxy software), in which case they are omitted in Table 5 since RD and PW could not reflect the efficiency of the current collaboration. We will further discuss how we identified the legitimate assets in §5.4.

Shown in Column 2 of Table 5, before we started collaborating with the web app providers, 96% WAE malware Marsea analyzed still had live assets on the abused web app platform. Column 5 shows that assets on Website Builders (i.e., WordPress, Tripod, Webs) are particularly long-lived. For example, Marsea identified the Symmi malware abusing the WordPress website `turrona5.wordpress.com` to resolve the next-hop endpoint. Surprisingly, the content on the website was posted in 2011 and is still live until 2022. Since we took down these assets by collaborating with web app providers (discussed in §5.5), it is evident that IRs failed to initialize the collaboration. Worse still, the Summary row of Table 5 shows that the RD of IRs reaches 253 days on average. In other words, analyzed WAE malware could go undetected for approximately 253 days. More interestingly, Column 10 shows the average PW (μ_{pw}) for each abused asset. A positive μ_{pw} (e.g., Github) indicates that the abused assets were updated even after the malware was detected.

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

5.4 WAE Malware Engagement Data

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

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

The engagement data results are presented in Table 6. Columns 4-5 show abused assets' First and Last Creation Date (FCD and LCD). Column 6 shows the geo-location (Geo) or language (Lan) of the abused assets derived using polygot [44]. Column 7 presents the engagement data supported by each web app. Rows 1-5 of Table 6 present the 5 Pastebin users and posts used by WAE malware, from which Marsea harvested the view number of the posts (Visits). Surprisingly, Marsea uncovered 908,109 views from 5 posts, averaging 181,621 views per post, indicating a significant extent of infection.

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

From Rows 6-18 of Table 6, we made several observations: (1) Column 7 shows that the average LoC of the README of benign repos and malicious repos are 91 and 1, respectively. Unlike benign projects that use README files, malicious repos do not require such comprehensive information. (2) Rows 6-16 show that the average information completeness ($Info$) of identities that create benign repositories and malicious repositories are 82.88% and 73.3%, respectively. Understandably, attackers prefer to leave this information blank during registration. Interestingly, Rows 13-16 of Table 6 show that attackers created 4 different GitHub repos on the same day, which the same malware family abused. We suspect that weak registration verification encourages attackers to create multiple assets using different identities.

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

Family	Identity	Asset	*FCD*	*LCD*	Lan/Geo	Engagement		
Pastebin (Asset: Post)								
Neshta	vinmarcio	g3w5Zkzi	2022-01-01	2022-01-01	Alabanian	AllAssets: 3 AllViews: 6,142	Types: TXT(3) Visits: 6,099	AllFCD: 2022-01-01
Bymeria	Huynhnhi92	Y8VWhxtG	2021-07-22	2021-07-22	English	AllAssets: 5 AllViews: 426,979	Types: TXT(3) Types:AutoIT(2)	AllFCD: 2019-09-19 Visits: 108,629
Ursu	pastwahman	xkkQYFEa	2018-01-09	2018-01-09	English	AllAssets: 47 AllViews: 5,389,380	Types: TXT(47) Visits: 793,381	AllFCD: 2016-11-11
Wacatac	sdfgsdf	qiMfPtmr	2021-12-21	2021-12-21	Russian	Visits: 3866		
Zusy	anthrax_	QKAdpkY	2021-12-19	2021-12-19	English	Visits: 796		
Github (Asset: Repo)								
Aauto	XTLS	Xray-core[1]	2020-11-09	2022-05-25	English	README: 99 Contributors: 39	Star: 8,499 Repos: 17	Fork: 1,448 Info: 84%
Apost	clangremlini	ayeloader-dll-repo[1]	2020-04-16	2022-02-03	English	README: 24 Contributors: 2	Star: 4 Repos: 13	Fork: 3 Info: 81%
Ayajbrci	kevlu8	OpenFRE[1]	2022-01-16	2022-03-28	Canada	README: 13 Star: 3	Followers: 12 Repos: 50	Fork: 2 Info: 90.6%
Msilzilla	fireworm71	veriumMiner[1]	2014-04-23	2022-01-25	English	README: 274 Info: 71.8%	Star: 59 Followers: 8	Fork: 38 Repos: 4
Onlinegames	blaumaus	le_chiffre[1]	2020-11-14	2022-04-03	Ukraine	README: 45 Info: 87.5%	Star: 178 Followers: 35	Fork: 27 Repos: 32
Redcap	maks88192939	Programm	2020-09-28	2020-09-28	English	README: 0 Info: 71.8%	Followers: 49 Repos: 32	
Lazy	0x00BAD	st_mod	2020-05-25	2022-01-28	English	README: 0 Info: 78%	Fork: 2 Repos: 4	
Wacatac	aaleaf	Lite	2020-04-10	2022-11-16	English	README: 0 Info: 78%	Fork: 0 Repos: 9	
Tasker	max444432	RMS2	2021-09-09	2022-07-14	English	README: 1 Info: 71.8%	Fork: 0 Repos: 12	
Growtopic	aphony12	imcool	2021-06-27	2021-09-20	English	README: 2 Contributors: 1	Repos: 1 Info: 71.8%	
	deviluker	jszpy	2021-06-27	2021-06-27	English	README: 2 Contributors: 1	Repos: 1 Info: 71.8%	
	giganiga	nothing	2021-06-27	2021-09-20	English	README: 2 Contributors: 1	Repos: 2 Info: 71.8%	
	rosen6	rosesareroses	2021-06-27	2021-06-27	English	README: 2 Contributors: 1	Repos: 1 Info: 71.8%	
Telegram (Asset: Channel)								
Razy	h_money_1	h+money+1	2020-09-29	2020-09-29	English	Subscribers: 3	Views: 12	
Zenpak	johnyes13	johnyes1121	2020-12-20	2020-12-20	English	Subscribers: 3	Views: 6	
GDrive (Asset: File)								
Onlinegames	sahnoila24	hwid	2021-10-17	2021-10-17	English	Text: 76 Bytes		
Avemaria	reubzjohn	Egcs	2020-08-26	2020-08-26	Unknown	Binary: 574KB		
Netwiredrc	moham[2]	Aqzm	2020-04-03	2020-04-03	Unknown	Binary: 815KB		
Remcos	Lorenz Kraft Formal D.O.O.	Bftv Ortl	2020-04-23 2020-07-13	2020-04-23 2020-07-13	Unknown Unknown	Binary: 378KB Binary: 1MB		
Sabsik	yuser20	XeroTool.a3x	2021-10-24	2021-10-24	Unknown	Binary: 85KB		
Blogspot (Asset: Web Page)								
Sisron	Nemorent	Blog Post	2012-02-05	2014-01-12	Egypt	Blogs: 230	Profile Views: 535	
Vobfus	nmournt	Blog Post	2012-02-05	2014-01-01	Egypt	Blogs: 116	Profile Views: 535	
Zpevdo	freechecking911	Blog Post	2018-07-29	2018-11-08	Thai	Blogs: 118	Profile Views: 14	
Virut	togpage	Blog Post	2012-12-09	2014-03-11	Egypt	Blogs: 2,380	Profile Views: 667	
Wordpress (Asset: Web Page)								
Symmi	turrona5	Blog Post	2011-05-14	2011-05-14	English	Blogs: 1		
Sysn	negragarchada	Blog Post	2011-07-01	2011-07-01	English	Blogs: 1		
Ayajbrci	tenah[3]	Blog Post	2017-07-26	2017-07-26	Turkish	Blogs: 1		

1: Benign repositories hosting general-purpose programs that are fetched by WAE malware. See §5.4 for details. All other assets were manually verified as hosting malicious content. **2:** Short for mohamedshokry19792016. **3:** Short for tenahukame6823496829.

Table 6: WAE Engagement Information.

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

Table 7: Reporting To Web App Providers.

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

5.5 Novel Collaboration

The end goal of IR and web app collaboration is the remediation of web app abuse. During our study, Marsea identified 129 active malicious assets directly abused by 439 WAE malware. Using the proof-of-abuse generated by Marsea, we collaborated with web app providers who have taken down 65 malicious assets (50%) to date. The detailed results of these collaborations are shown in Table 7. Unfortunately, as shown in Rows 2 and 9, GitHub and Google took 103 and 138 days to take action. Consequently, only 12 of the 69 live assets reported to GitHub and Google have been remediated. Of the remaining 60 reported assets on other web apps, 53 (83%) were taken down within 2 weeks. We continue to pursue the remediation of the remainder.

We categorize the collaboration into *bilateral* and *unilateral* in Column 1 of Table 7. Bilateral means web app providers update IRs throughout the remediation process and provide IRs with a response. This allows IRs to track the remediation process and assist web app providers when necessary. Unilateral means that IRs can only report the asset/identity as abusive via a ticket system. Reporting tools are often as simple as a button on the web app (e.g., Google does not provide any way for IRs to provide proof-of-abuse). Consequently, IRs are excluded from the subsequent remediation process. The directly abused assets we reported and took down are shown in Columns 4-5. Interested readers are directed to Appendix B for responses we received for bilateral collaboration. From Table 7, we observe the lengthy response period by more popular and reputable web app providers. For example, Google took 138 days to react. Even though unilateral collaboration can remediate the abuse relatively quickly, it can easily create hurdles for follow-up actions initialized by IRs if the remediation is slow. Therefore, rapid bilateral collaboration should be encouraged.

Lateral Remediation. As shown in Rows 1-3 of Table 6, besides the 3 posts directly abused by WAE malware, Marsea found 52 more posts from these identities. These additional assets give us a unique opportunity to perform lateral remediation during our novel collaboration. As shown in Row 3 of Table 6, Marsea found the Ursu malware using the post (with paste key `xkkQYFEa`) by `pastwahman` on Pastebin. Enabled by the Pastebin API, Marsea can locate another 46 posts from `pastwahman`. The earliest creation date spans 424 days before the FCD of the directly abused post. Marsea found that these potentially malicious posts have 5,822,501 views, 4,909,908 more than the directly abused posts. We collaborated with Pastebin to remove these assets from the same identity. Surprisingly, during our investigation of all posts from the `pastwahman` identity, we found the `DARKOM` (paste key `kE8tbd6t`) post with a VB script. The script queried a Pastebin post named `allinone` (paste key `q6dVaKSf`) under a different identity, `Getsilent`. Through further collaboration with Pastebin to remediate `Getsilent`, we took down 30 additional posts. Overall, starting from 1 post, lateral collaboration enabled us to take down an additional 76 assets.

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

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

1,2,3: Short for itroblvehacker.gq, Darkkomet, Bladabindi.
4: Presents the payload dropping.

Table 8: Multi-Stage Abuse Chain.

use ↳ to present the dropper chain.

From Table 8, we observed that from 14 first-stage WAE malware (from 3 families), Marsea revealed 15 additional dropped malware from 6 families. More importantly, as shown in Column 8 of Table 8, the initial infections only abuse web apps (0% in the %M column). Later-stage malware increasingly rely on attacker-controlled C&C servers as the abuse chain gets longer (ending at 100% in the %M column). Ensuring the initial stage remains stealthy and persistent allows the attackers to update the assets on the web apps to drop new payloads once the later-stage malware are flagged as malicious. It is also interesting to note that 38% of malware dropped by GDrive connects back to GDrive to download different next-stage payloads, seeking persistent infection. This suggests a malware development toolkit abusing GDrive and Discord.

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

Migration Remediation. Marsea observed 4 cases of WAE malware migrating across different web apps or different identities under the same web app. We hand-verified the content of the abused assets, and if the abused assets share the same content or format, we use it as proof of the migration. The results are shown in Table 9.

Family	Mal	*FSD*	Web Apps	Vector	Iden	Assets
Sohanad	2 1	2013-06 2013-08	Webs Webs	FD FD	`se***3` `ad***9`	`settigs.xls` `settings.doc`
Sabsik	1 8	2021-06 2022-10	GitHub GitHub	MF MF	`cu***u` `Ar***k`	`uattached` `auth`

Hynamer	2
	1
		2021-05
		2021-09
			Github
			Discord
				EXE
				EXE
					`sw**a`
					`88***2`
						`Nsudo.exe`
						`Nsudo.exe`

Msil	1
	1
		2022-01
		2022-01
			Dropbox
			GDrive
				MF
				MF
					`4g****a`
					`sa****a`
						`hwid.txt`
						`hwid.txt`

Table 9: WAE Malware Migration.

From Table 9, we made the following observations: (1) WAE migration can be divided into intra- and inter-web apps, as shown in Rows 1-4 and 5-8, respectively. (2) 100% of WAE malware in this case study changed identity names during migration. (3) 2 malware families (shown in Row 5-8), which account for 29.4% of the malware samples in Table 9, keep the same asset name during the migration. The attackers only update the assets name during intra-web app migration Therefore, an information-sharing process among web app providers is needed to enable content-based asset monitoring.

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

6 Related Work

Empirical Study of WAE Malware. Past research studied WAE as seen from the botnet [16], [18], [19], [45], [46] by detailing one or several specific attacks. Barroso et al. [47] presented a survey of cybercriminals exploiting public web servers to carry out different attacks. Zhang et al. [48] proposed an approach to detect compromised websites for searching poisoning attacks. Wang et al. [17] performed a comprehensive analysis of Economic Denial of Sustainability (EDoS) attacks conducted through free public third-party services. Liao et al. [20] conducted a study on the abuse of cloud repositories as a malicious service by analyzing redirection chains. Li et al. [49] studied abusive hosts on Internet Service Providers (ISP). Alrwais et al. [50] studied the malicious activities which were hosted on the sub-allocations of the legitimate service provider. Botacin et al. [27] performed a longitudinal analysis of

Brazilian Financial Malware, revealing the abuse of web apps. Han et al. [46] performed a large-scale analysis of malware interacting with Amazon EC2 [51]. Lever et al. [52] used the malware execution traces and the DNS queries to study the efficacy of network detection. While these studies focused on different attacks and analyzed cloud services, Marsea analyzes WAE malware to support live actions against malware. Our approach goes beyond investigating web apps to reveal unexpected abuse of non-standard web apps.

Malware Behavior Analysis. There are works using taint analysis [53], [54], API trace analysis [55]–[60], and network traffic analysis [61]–[63] to reveal malware behaviors. However, Marsea uses symbolic data propagation to track the information flow and to perform multi-path exploration. Marsea bridges the gap between analyzed WAE malware and the abused web apps by attributing malicious vectors to the abused identities and assets. Further, Marsea harvests the engagement information of abused identities and assets to evaluate the extent of the infection.

Concolic Analysis. Several tools perform concolic analysis on a source code level [64]–[66] and a binary level [35], [67]–[70]. Since these tools suffer from path explosion, the research community has several strategies to mitigate this problem [71], [72]. However, these tools are largely geared toward vulnerability analysis. There are also existing works built on top of S2E [37], [38], [73], [74]. Unfortunately, these works serve different goals from WAE malware analysis. Instead, Marsea uses a new code-orientated exploration strategy and introduces symbolic data through peripheral inputs and evasive functions. This enables Marsea to categorize WAE through data flow analysis and will not face the challenge introduced by context-sensitive malware or logic bombs.

7 Discussion and Limitations

Ethical Discussion. We remediated WAE malware by collaborating with the web app providers. During the remediation process, we strictly followed the guidelines and terms of service provided by web app providers (e.g. [75]–[80]) We *do not* store or use private credential information extracted from the malware to access any remote endpoints. We *do not* attempt to remediate the identities and assets WAE malware used without collaboration from web app providers.

Challenges of Concolic Analysis on Malware. Marsea is built upon concolic analysis. Like other concolic and symbolic analysis techniques [68], [81]–[83], Marsea can suffer from the path explosion motivating our design in §3.1. Our validation in §4 empirically shows that Marsea avoids path explosion in WAE malware. Furthermore, malware often employ packing or obfuscation techniques. Since Marsea uses concolic analysis, the malware can unpack itself at runtime [29]. Prior works [60], [70], [84], [85] explored handling sophisticated malware deploying code obfuscation specifically aiming to thwart symbolic analysis. We followed their best practices while designing Marsea.

8 Conclusion

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

9 Acknowledgement

We thank the anonymous reviewers for their constructive comments and feedback. In particular, we thank our shepherd for their guidance throughout the revision process. 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 N00014-18-1-2662; the Defense Advanced Research Projects Agency (DARPA) under contract N66001-21-C-4024; the National Science Foundation (NSF) under grant 2143689; and Cisco Systems under an unrestricted gift. 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] *Cloud CDN: Content Delivery Network*, https://cloud.google.com/cdn, [Accessed: 2022-04-05].

[2] *Azure Content Delivery Network*, https://azure.microsoft.com/en-us/services/cdn/, [Accessed: 2022-04-05].

[3] *Content Delivery Network (CDN) - Amazon CloudFront*, https://aws.amazon.com/cloudfront/, [Accessed: 2022-04-05].

[4] *GitHub*, https://github.com/, [Accessed: 2022-04-15].

[5] *Pastebin.com - #1 paste tool since 2002!* https://pastebin.com/, [Accessed: 2021-07-22].

[6] *Dropbox*, https://www.dropbox.com/, [Accessed: 2021-04-06].

[7] *Google Drive*, https://www.drive.google.com/, [Accessed: 2021-04-06].

[8] *Amazon S3*, https://aws.amazon.com/s3/, [Accessed: 2021-04-06].

[9] *Twitter*, https://twitter.com/, [Accessed: 2021-04-06].

[10] *Instragram*, https://www.instagram.com/, [Accessed: 2021-04-06].

[11] *Telegram Messenger*, https://telegram.org/, [Accessed: 2021-04-06].

[12] *Turla Hacking Group Launches New Backdoor in Attacks Against US, Afghanistan*, https://www.zdnet.com/article/turla-hacking-group-launches-new-backdoor-in-attacks-against-us-afghanistan/, [Accessed: 2022-04-05].

[13] *Russian Hacking Group Uses Dropbox to Store Malware-stolen Data*, https://www.bleepingcomputer.com/news/security/russian-hacking-group-uses-dropbox-to-store-malware-stolen-data, [Accessed: 2022-04-05].

[14] *Turla's Crutch Backdoor Leverages Dropbox in Espionage Attacks*, https://threatpost.com/turla-backdoor-dropbox-espionage-attacks/161777/, [Accessed: 2022-04-05].

[15] *Iranian Hackers Abuse Slack For Cyber Spying*, https://www.forbes.com/sites/thomasbrewster/2021/12/15/iran-backed-hackers-use-slack-for-cyber-espionage, [Accessed: 2022-04-05].

[16] H. Badis, G. Doyen, and R. Khatoun, "Understanding botclouds from a system perspective: A principal component analysis," in *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*, IEEE, 2014.

[17] H. Wang, Z. Xi, F. Li, and S. Chen, "Abusing public third-party services for edos attacks," in *Proceedings of the 10th USENIX Workshop on Offensive Technologies (WOOT)*, Austin, TX, Aug. 2016.

[18] N. Pantic and M. I. Husain, "Covert Botnet Command and Control Using Twitter," in *Proceedings of the 31st Annual Computer Security Applications Conference (ACSAC)*, Los Angeles, CA, Dec. 2015.

[19] G. Lingam, R. R. Rout, D. V. L. N. Somayajulu, and S. K. Das, "Social Botnet Community Detection: A Novel Approach based on Behavioral Similarity in Twitter Network using Deep Learning," in *Proceedings of the 15th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, Taipei, Taiwan, Oct. 2020.

[20] X. Liao, S. Alrwais, K. Yuan, *et al.*, "Cloud repository as a malicious service: Challenge, identification and implication," *Cybersecurity*, vol. 1, no. 1, p. 14, 2018.

[21] S. Gheewala and R. Patel, "Machine learning based twitter spam account detection: A review," in *2018 Second International Conference on Computing Methodologies and Communication (ICCMC)*, IEEE, 2018, pp. 79–84.

[22] S. Lee and J. Kim, "Warningbird: A near real-time detection system for suspicious urls in twitter stream," *IEEE Transactions on Dependable and Secure Computing*, vol. 10, no. 3, pp. 183–195, 2013.

[23] A. Sanzgiri, A. Hughes, and S. Upadhyaya, "Analysis of malware propagation in twitter," in *2013 IEEE 32nd International Symposium on Reliable Distributed Systems*, 2013, pp. 195–204.

[24] F. Concone, A. De Paola, G. L. Re, and M. Morana, "Twitter analysis for real-time malware discovery," in *2017 AEIT International Annual Conference*, 2017, pp. 1–6.

[25] V. Radunović and M. Veinović, "Malware command and control over social media: Towards the server-less infrastructure," *SJEE*, vol. 17, no. 3, pp. 357–375, 2020.

[26] S. Miller and P. Smith, "Rise of legitimate services for backdoor command and control," Anomali, Tech. Rep., 2017.

[27] M. Botacin, H. Aghakhani, S. Ortolani, *et al.*, "One size does not fit all: A longitudinal analysis of brazilian financial malware," *ACM Trans. Priv. Secur.*, vol. 24, no. 2, Jan. 2021.

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

[29] J. Fuller, R. P. Kasturi, A. Sikder, *et al.*, "C3PO: Large-Scale Study of Covert Monitoring of C&C Servers via Over-Permissioned Protocol Infiltration," in *Proceedings of the 28th ACM Conference on Computer and Communications Security (CCS)*, Seoul, South Korea, Nov. 2021.

[30] C. Rossow, D. Andriesse, T. Werner, *et al.*, "Sok: P2pwned - modeling and evaluating the resilience of peer-to-peer botnets," in *Proceedings of the 34th IEEE Symposium on Security and Privacy (S&P)*, San Francisco, CA, May 2013, pp. 97–111.

[31] B. B. Kang, E. Chan-Tin, C. P. Lee, *et al.*, "Towards Complete Node Enumeration in a Peer-to-peer Botnet," in *Proceedings of the 4th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, Sydney, Australia, Mar. 2009, pp. 23–34.

[32] *MITRE ATT&CK Web Service*, https://attack.mitre.org/techniques/T1102/, [Accessed: 2021-04-21].

[33] *Malpedia: Free and Open Malware Reverse Engineering Resource offered by Fraunhofer FKIE*, https://malpedia.caad.fkie.fraunhofer.de, [Accessed: 2021-05-02].

[34] *Web Service, Technique T1102 - Enterprise | MITRE ATT&CK®*, https://attack.mitre.org/techniques/T1102/, [Accessed: 2020-09-19].

[35] V. Chipounov, V. Kuznetsov, and G. Candea, "S2E: A Platform for In-vivo Multi-path Analysis of Software Systems," *ACM SigPlan Notices*, vol. 46, no. 3, pp. 265–278, 2011.

[36] Y. Hu, G. Huang, and P. Huang, "Automated reasoning and detection of specious configuration in large systems with symbolic execution," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 719–734.

[37] A. Altinay, J. Nash, T. Kroes, *et al.*, "Binrec: Dynamic binary lifting and recompilation," in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–16.

[38] S. Walla and C. Rossow, "Malpity: Automatic identification and exploitation of tarpit vulnerabilities in malware," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE, 2019, pp. 590–605.

[39] S. Sebastián and J. Caballero, "AVclass2: Massive Malware Tag Extraction from AV Labels," in *Proceedings of the 36th Annual Computer Security Applications Conference (ACSAC)*, Virtual Conference, Dec. 2020, pp. 42–53.

[40] *VirusTotal - Intelligence Search*, https://developers.virustotal.com/reference/intelligence-search, [Accessed: 2022-10-01].

[41] O. Alrawi, C. Lever, K. Valakuzhy, K. Snow, F. Monrose, M. Antonakakis, *et al.*, "The circle of life: A large-scale study of the iot malware lifecycle," in *Proceedings of the 30th USENIX Security Symposium (Security)*, Virtual Conference, Aug. 2021.

[42] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen, "Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation," in *Proceedings of the 2019 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2019.

[43] *Cisco Talos Intelligence Group - Comprehensive Threat Intelligence*, https://www.talosintelligence.com, [Accessed: 2022-04-15].

[44] M. A. Islam, A. B. M. A. A. Islam, and M. S. H. Anik, "Polygot: An approach towards reliable translation by name identification and memory optimization using semantic analysis," in *2017 4th International Conference on Networking, Systems and Security (NSysS)*, 2017, pp. 1–8.

[45] K. Clark, M. Warnier, and F. M. Brazer, "The future of cloud-based botnets?" In *Proceedings of the 1st International Conference on Cloud Computing and Services Science (CLOSER)*, Noordwijkerhout, Netherlands, May 2011.

[46] X. Han, N. Kheir, and D. Balzarotti, "The role of cloud services in malicious software: Trends and insights," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, 2015, pp. 187–204.

[47] D. Barroso, "Botnets - the silent threat," *European Network and Information Security Agency (ENISA)*, vol. 15, p. 171, 2007.

[48] J. Zhang, C. Yang, Z. Xu, and G. Gu, "Poisonamplifier: A guided approach of discovering compromised websites through reversing search poisoning attacks," in *Proceedings of the 15th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, Amsterdam, Netherlands, Sep. 2012.

[49] Z. Li, S. Alrwais, Y. Xie, F. Yu, and X. Wang, "Finding the linchpins of the dark web: A study on topologically dedicated hosts on malicious web infrastructures," in *Proceedings of the 34th IEEE Symposium on Security and Privacy (S&P)*, San Francisco, CA, May 2013.

[50] S. Alrwais, K. Yuan, E. Alowaisheq, Z. Li, and X. Wang, "Understanding the dark side of domain parking," in *Proceedings of the 22th USENIX Security Symposium (Security)*, Washington, DC, Aug. 2013.

[51] *Amazon EC2 - Secure Cloud Services*, https://aws.amazon.com/ec2/, [Accessed: 2022-10-01].

[52] C. Lever, P. Kotzias, D. Balzarotti, J. Caballero, and M. Antonakakis, "A Lustrum of Malware Network Communication: Evolution and Insights," in *Proceedings of the 38th IEEE Symposium on Security and Privacy (S&P)*, San Jose, CA, May 2017, pp. 788–804.

[53] L. Martignoni, E. Stinson, M. Fredrikson, S. Jha, and J. C. Mitchell, "A Layered Architecture for Detecting Malicious Behaviors," in *Proceedings of the 11th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, Cambridge, Massachusetts, Sep. 2008, pp. 78–97.

[54] E. Stinson and J. C. Mitchell, "Characterizing Bots' Remote Control Behavior," in *Proceedings of the Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, Lucerne, CH, Jul. 2007, pp. 89–108.

[55] E. Kirda, C. Kruegel, G. Banks, G. Vigna, and R. Kemmerer, "Behavior-based Spyware Detection," in *Proceedings of the 15th USENIX Security Symposium (Security)*, Vancouver, Canada, Jul. 2006, p. 694.

[56] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X.-y. Zhou, and X. Wang, "Effective and Efficient Malware Detection at the End Host," in *Proceedings of the 18th USENIX Security Symposium (Security)*, vol. 4, Montreal, Canada, Aug. 2009, pp. 351–366.

[57] G. J. Széles and A. Coleşa, "Malware Clustering Based on Called API During Runtime," in *Proceedings of the International Workshop on Information and Operational Technology and Security (IOSec)*, Crete, GR, Sep. 2018, pp. 110–121.

[58] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, "Copperdroid: Automatic reconstruction of android malware behaviors," in *Proceedings of the 2015 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2015.

[59] A. Lanzi, M. I. Sharif, and W. Lee, "K-tracer: A system for extracting kernel malware behavior," in *Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2009.

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

[61] X. Deng and J. Mirkovic, "Malware Analysis Through High-level Behavior," in *Proceedings of the 11th USENIX Workshop on Cyber Security Experimentation and Test (CSET)*, Baltimore, MD, Aug. 2018.

[62] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, and C. Kruegel, "A View on Current Malware Behaviors," in *Proceedings of the 2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, Boston, MA, Apr. 2009.

[63] S. Wang, Q. Yan, Z. Chen, B. Yang, C. Zhao, and M. Conti, "Detecting android malware leveraging text semantics of network flows," *IEEE Transactions on Information Forensics and Security*, 2018.

[64] P. Godefroid, N. Klarlund, and K. Sen, "DART: Directed automated random testing," in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, Chicago, IL, Jun. 2005.

[65] C. Cadar, V. Ganesh, P. M. Pawlowski, D. L. Dill, and D. R. Engler, "EXE: Automatically generating inputs of death," *ACM Transactions on Information and System Security*, 2008.

[66] C. Cadar, D. Dunbar, D. R. Engler, *et al.*, "Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs," in *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, San Diego, CA, Dec. 2008.

[67] P. Godefroid, M. Y. Levin, D. A. Molnar, *et al.*, "Automated whitebox fuzz testing," in *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2008.

[68] Y. Shoshitaishvili, R. Wang, C. Salls, *et al.*, "Sok:(state of) the art of war: Offensive techniques in binary analysis," in *Proceedings of the 37th IEEE Symposium on Security and Privacy (S&P)*, San Jose, CA, May 2016.

[69] S. K. Cha, T. Avgerinos, A. Rebert, and D. Brumley, "Unleashing Mayhem on Binary Code," in *Proceedings of the 33rd IEEE Symposium on Security and Privacy (S&P)*, San Francisco, CA, May 2012, pp. 380–394.

[70] L. Martignoni, S. McCamant, P. Poosankam, D. Song, and P. Maniatis, "Path-exploration lifting: Hi-fi tests for lo-fi emulators," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 1, pp. 337–348, Mar. 2012.

[71] E. Bounimova, P. Godefroid, and D. Molnar, "Billions and billions of constraints: Whitebox fuzz testing in production," in *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, San Francisco, CA, May 2013.

[72] I. Haller, A. Slowinska, M. Neugschwandtner, and H. Bos, "Dowsing for overflows: A guided fuzzer to find buffer boundary violations," in *Proceedings of the 22th USENIX Security Symposium (Security)*, Washington, DC, Aug. 2013.

[73] M. N. Alsaleh, J. Wei, E. Al-Shaer, and M. Ahmed, "Gextractor: Towards automated extraction of malware deception parameters," in *Proceedings of the 8th Software Security, Protection, and Reverse Engineering Workshop*, 2018, pp. 1–12.

[74] S. Y. Kim, S. Lee, I. Yun, *et al.*, "Cab-fuzz: Practical concolic testing techniques for cots operating systems," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, 2017, pp. 689–701.

[75] *GitHub Terms of Service*, https://docs.github.com/en/site-policy/github-terms/github-terms-of-service, [Accessed: 2022-10-01].

[76] *Report Abuse*, https://support.github.com/contact/report-abuse, [Accessed: 2022-10-01].

[77] *Discord's Terms of Service*, https://discord.com/terms, [Accessed: 2022-10-01].

[78] *Discord's Request Submission*, https://dis.gd/request, [Accessed: 2022-10-01].

[79] *Pastebin.com Terms of Service*, https://pastebin.com/doc_terms_of_service, [Accessed: 2022-10-01].

[80] *Pastebin Report*, https://pastebin.com/contact, [Accessed: 2022-10-01].

[81] D. Brumley, C. Hartwig, M. G. Kang, *et al.*, "Bitscope: Automatically dissecting malicious binaries," CS-07-133, School of Computer Science, Carnegie Mellon University, Tech. Rep., 2007.

[82] Y. Li, Z. Su, L. Wang, and X. Li, "Steering symbolic execution to less traveled paths," *ACM SigPlan Notices*, vol. 48, no. 10, pp. 19–32, 2013.

[83] V. Kuznetsov, J. Kinder, S. Bucur, and G. Candea, "Efficient state merging in symbolic execution," *Acm Sigplan Notices*, vol. 47, no. 6, pp. 193–204, 2012.

[84] U. Bayer, A. Moser, C. Kruegel, and E. Kirda, "Dynamic analysis of malicious code," *Journal in Computer Virology*, vol. 2, no. 1, pp. 67–77, 2006.

[85] A. Moser, C. Kruegel, and E. Kirda, "Exploring multiple execution paths for malware analysis," in *2007 IEEE Symposium on Security and Privacy (SP'07)*, IEEE, 2007, pp. 231–245.

A Web Apps And Available Information

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

1: Creation Time (CT) and Update Time (UT).
2: Other assets from the same identity.

Table 10: Web Apps And Available Information

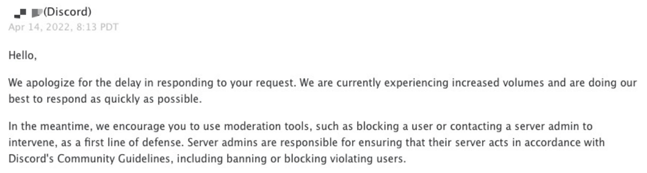
Table 10 presents the web apps Marsea found during our study and the information publicly available from each web app. Columns 1-2 present the web app category and the corresponding web apps. Columns 3-4 present the identity and assets Marsea supported to identify. Columns 5-8 show whether the web apps expose asset Creation Time (CT), Update Time (UT), other assets from the same identity (OA), and the views of the assets, respectively.

Please note that as stated in §5.2, Local Info web apps shown in Column 1 of Table 4 have no concept of identity and assets, thus, are not included in Table 10. Interestingly, as shown in Row Rows 10-11 of Table 10, Marsea found that WAE malware abuse webhook. However, Slack does not expose information to the public, given only a webhook. Unlike Slack, as shown in Row 11 of Table 10, Discord revealed the channel ID in the webhook endpoint, through which Marsea could harvest *CT* and *UT*. As shown in Columns 7-8, only 10 (50%) and 4 (20%) web apps expose the OA and views of assets to the public. Given the importance of OA and views of assets when IRs and web app providers pursue the *Lateral Remediation* (§5.3) and extent of infection evaluation, making these data publicly available or at the early stage of collaboration should be encouraged.

Meanwhile, given Marsea's ability to identify vectors and attribute sessions to them, IRs could still collaborate with the web app providers.

B Response From Web App Providers

Figure 2 presents our communication with Discord when they experienced high reporting volume, and Figure 3 shows the takedown confirmation by Discord. Figure 5 shows the takedown confirmation by MediaFire. Figure 4 shows the case where GitHub needed 103 days to initialize the investigation.



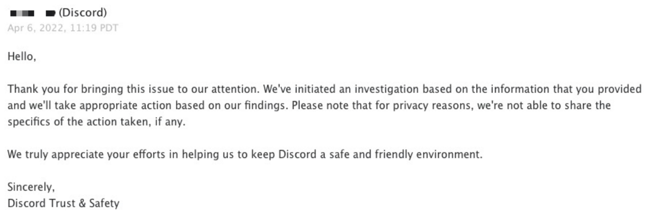
[redacted] (Discord)
Apr 14, 2022, 8:13 PDT

Hello,

We apologize for the delay in responding to your request. We are currently experiencing increased volumes and are doing our best to respond as quickly as possible.

In the meantime, we encourage you to use moderation tools, such as blocking a user or contacting a server admin to intervene, as a first line of defense. Server admins are responsible for ensuring that their server acts in accordance with Discord's Community Guidelines, including banning or blocking violating users.

Figure 2: Late Response From Discord.



[redacted] (Discord)
Apr 6, 2022, 11:19 PDT

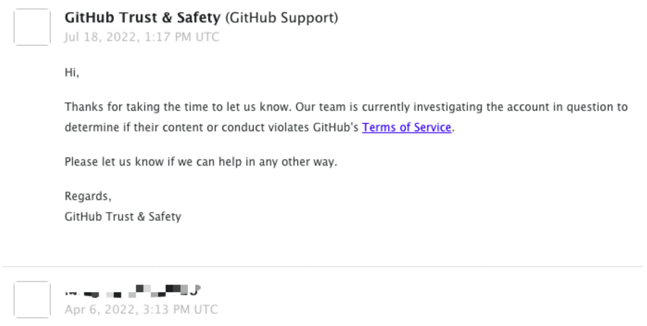
Hello,

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

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

Sincerely,
Discord Trust & Safety

Figure 3: Confirmation By Discord.



GitHub Trust & Safety (GitHub Support)
Jul 18, 2022, 1:17 PM UTC

Hi,

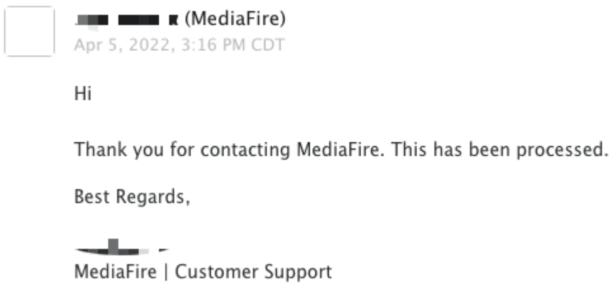
Thanks for taking the time to let us know. Our team is currently investigating the account in question to determine if their content or conduct violates GitHub's Terms of Service.

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

Regards,
GitHub Trust & Safety

[redacted]
Apr 6, 2022, 3:13 PM UTC

Figure 4: Response From GitHub.



[redacted] (MediaFire)
Apr 5, 2022, 3:16 PM CDT

Hi

Thank you for contacting MediaFire. This has been processed.

Best Regards,

[redacted]
MediaFire | Customer Support

Figure 5: Response From MediaFire.