Comparing Malware Evasion Theory with Practice: Results from Interviews with Expert Analysts

Miuyin Yong Wong, Matthew Landen, Frank Li, Fabian Monrose, and Mustaque Ahamad, *Georgia Institute of Technology*

https://www.usenix.org/conference/soups2024/presentation/yong-wong

This paper is included in the Proceedings of the Twentieth Symposium on Usable Privacy and Security.

August 12–13, 2024 • Philadelphia, PA, USA

978-1-939133-42-7

Open access to the Proceedings of the Twentieth Symposium on Usable Privacy and Security is sponsored by USENIX.

Comparing Malware Evasion Theory with Practice: Results from Interviews with Expert Analysts

Miuyin Yong Wong
Georgia Institute of Technology

Matthew Landen
Georgia Institute of Technology

Frank Li
Georgia Institute of Technology

Fabian Monrose
Georgia Institute of Technology

Mustaque Ahamad
Georgia Institute of Technology

Abstract

Malware analysis is the process of identifying whether certain software is malicious and determining its capabilities. Unfortunately, malware authors have developed increasingly sophisticated ways to evade such analysis. While a significant amount of research has been aimed at countering a spectrum of evasive techniques, recent work has shown that analyzing malware that employs evasive behaviors remains a daunting challenge. To determine whether gaps exist between evasion techniques addressed by research and challenges faced by practitioners, we conduct a systematic mapping of evasion countermeasures published in research and juxtapose it with a user study on the analysis of evasive malware with 24 expert malware analysts from 15 companies as participants. More specifically, we aim to understand (i) what malware evasion techniques are being addressed by research, (ii) what are the most challenging evasion techniques malware analysts face in practice, (iii) what are common methods analysts use to counter such techniques, and (iv) whether evasion countermeasures explored by research align with challenges faced by analysts in practice. Our study shows that there are challenging evasion techniques highlighted by study participants that warrant further study by researchers. Additionally, our findings highlight the need for investigations into the barriers hindering the transition of extensively researched countermeasures into practice. Lastly, our study enhances the understanding of the limitations of current automated systems from the perspective of expert malware analysts. These contributions suggest new research directions that could help address the challenges posed by evasive malware.

1 Introduction

Malicious software or malware is a serious and constantly evolving threat to cybersecurity. As of 2023, a staggering 300,000 new malware samples are generated daily[1]. With such a large volume of new malware, it is imperative that security professionals are equipped with the tools necessary to identify and analyze these samples in a timely manner. Unfortunately, as malware becomes more sophisticated, malware authors have developed evasive techniques to make the analysis more difficult and time-consuming. In fact, in 2018 a study showed that 98% of malware samples employ at least one evasive technique[2]. A few examples of evasive techniques are code obfuscation and sandbox evasion [1, 28, 48].

Code obfuscation is a technique that deliberately makes the code more difficult to understand during static analysis, which is the process of examining a malware's functionality without executing the code. Some common examples of code obfuscation include string encryption, packing, flattening the control flow, and adding spurious code. In turn, to mitigate obfuscation techniques, researchers have developed unpackers [13, 17, 31, 49, 74, 82], and de-obfuscators [7, 18, 65, 73, 84, 94]. To impede dynamic analysis, a type of analysis that executes the malware in a controlled environment, malware authors often insert checks in their code to detect if they are being executed within an analysis environment such as a sandbox, a technique known as sandbox evasion. Such checks enable malware to evade analysis by not revealing its functionality. In response, researchers have developed more transparent sandboxes which make the analysis environment less detectable [20, 21, 26, 68, 80, 81, 87] and techniques to detect evasive samples by comparing multiple executions of the malware [30, 32, 40, 45, 53, 83]. More recently, there have been efforts such as forced execution [38, 58, 79], which aim to investigate the different execution paths of malware to expose its malicious behavior. Despite the considerable amount of research aimed at evasion techniques, a user study by Yong et al. [88] found that analyzing evasive malware continues to be a challenge for practitioners.

To keep up with the increased sophistication of evasive malware and explore how future research can enhance the analysis of evasive malware in practice, it is necessary to understand (i) methods that have been developed by past re-

[1] See 50+ Cybersecurity Statistics for 2023 You Need to Know

[2] See Evasive Malware Now a Commodity.

search to counter evasion, and (ii) the evasion techniques that still remain challenging for malware analysts in practice. Although prior work has conducted surveys of dynamic malware analysis evasion techniques [1, 10], none focused specifically on countermeasures that help handle evasion techniques that hinder either dynamic or static analysis. Additionally, while prior user studies [77, 88] have studied the process of reverse engineering and malware analysis, our study is the first to identify the specific evasion techniques that malware analysts in practice find challenging and examine how they currently handle such techniques. Moreover, unlike prior studies, we conduct a systematic comparison with existing literature to provide informed recommendations for future research that may help solve analysts' challenges. To meet this goal, we conduct the first systematic mapping of countermeasures for evasion techniques employed by malware, and combine it with 24 semi-structured interviews with highly experienced malware analysts who work in established security groups of well-known companies such as Proofpoint, General Electric (GE), Mandiant (now Google), IBM, and SecureWorks.

Our systematic mapping allows us to understand which malware evasion techniques have been addressed in research, and the methodologies that have been developed to counter such evasion techniques. Furthermore, our user study helps us identify evasion techniques that remain challenging in practice. Together, our systematic mapping and user study are used to perform a comparative analysis between the evasion countermeasures that research focuses on and the challenging evasive techniques practitioners encounter. This can also help inform areas in need of further research. Thus, we seek to answer the following questions:

RQ1 Which malware evasion techniques have been the focus of research dealing with evasion countermeasures?

RQ2 What are the most challenging evasive techniques encountered by malware analysts in practice?

RQ3 What approaches do malware analysts take to counter evasive techniques?

The main contributions of this paper are the following: First, we map and categorize evasive countermeasures found in the literature. Second, we identify the most challenging evasion techniques encountered by our study participants, along with the manual processes they follow to overcome such challenges. Third, we conduct a comparative analysis between solutions explored by research and challenges encountered by malware analysts in practice. Our analysis reveals that existing research solutions have significantly contributed to the field. However, there exists a misalignment between some of the practical challenges that malware experts face with evasive malware and the focus of developed research solutions. For example, we found that although malware analysts find anti-disassembly to be a significant hurdle, there is relatively less focus on research being done on anti-disassembly in the scope of malware analysis. Conversely, we found that despite the significant amount of research on countering obfuscation techniques, participants state obfuscation as the most challenging evasion technique to handle. These observations provide valuable insights for identifying future research directions, including the development of innovative tools to assist analysts with persistent challenges and the investigation of barriers hindering the transition of existing research techniques into practice.

2 Systematic Mapping Methodology

In this section, we introduce our systematic mapping of countermeasures for evasion techniques. While there are surveys of dynamic analysis evasion techniques [1, 10], they include limited information about their countermeasures. Furthermore, to the best of our knowledge, none have covered both static and dynamic analysis evasion countermeasures. Without an understanding of previous research efforts aimed at countering malware evasion techniques, research gaps in the field remain unclear. To fill this need and provide an overview of existing research on evasion countermeasures for both static and dynamic analysis, we conduct a systematic mapping.

We chose a systematic mapping approach because it systematically identifies knowledge gaps among existing research literature and uncovers promising future research directions within the field [60, 61]. More recently, this method has gained recognition in fields such as software engineering [2, 59] and medicine [11, 62], underscoring its effectiveness in enabling a rigorous and structured overview of the current research landscape. In this study, we followed Persons's [61] guidelines, which include formulating research questions, defining the search process, establishing clear inclusion and exclusion criteria, performing data extraction aligned with the research questions, and conducting data analysis.

2.1 Mapping Research Questions

The main objective of this mapping is to identify and analyze the solutions developed in research to counter evasion techniques for malware analysis. Specifically, our mapping aims to answer the following research questions:

MQ1 Which malware analysis evasion techniques have been addressed in research?

MQ2 What methodologies are proposed by researchers to counter evasion techniques?

2.2 Search Strategy

In this study, we chose a database search as our primary search strategy. Before starting our database search, the first and second authors conducted a manual search of relevant papers to identify keywords, necessary for the creation of our search query. The manual search began by reviewing the titles of

research papers from four top security conferences (USENIX, IEEE S&P, CCS, NDSS) published between 2012 and 2022 to identify papers related to the topic of malware analysis. We scope our systematic mapping to papers addressing Windows Exe malware that employ evasion techniques because Windows is the most targeted operating system by threat actors. In fact, in 2022 Mandiant reported that 92% of the newly identified malware families run on Windows[3]. Focusing solely on Windows Exe also allows us to provide a fair comparison between our participants' challenges and research on countermeasures discussed later in §8. Subsequently, the two authors applied the following predefined inclusion criteria to the titles and abstracts:

Inclusion Criteria on Title and Abstract.

- References dynamic malware analysis, deobfuscation, unpacking, or disassembly.
- Not aimed towards mobile or IoT malware.
- Not a survey or a measurement study.

After identifying 40 papers that satisfied the above inclusion criteria, both authors read the full text of the included papers and applied the following exclusion criteria:

Exclusion Criteria on Full Text.

- The research findings does not directly help counter either static or dynamic analysis evasion techniques such as anti-sandboxing, anti-debugging, obfuscation, or anti-disassembly, nor does it provide a way for the analysis to proceed without countering the evasion techniques.

In 8 cases where the two authors disagreed, the two authors reviewed the details of the paper together to resolve disagreements, resulting in a final set of 20 papers.

To construct the database search query, we first extracted keywords from the 20 papers identified through a manual search. To extract keywords, we applied common preprocessing steps to the abstracts, including lowercasing, removing special characters, and removing stop words. While stemming and lemmatization are also common preprocessing techniques, we opted not to utilize these techniques since we require the exact words to match during search queries and these techniques would not produce exact matches. After preprocessing the data, we categorized the abstracts based on the evasion technique they address, either dynamic or static, and utilized TF-IDF analysis to identify the most significant and frequently occurring words within each category. Following this, we categorized the remaining words based on their semantic commonalities. For instance, we grouped the words debugging, automatic, and dynamic together as they all relate to the execution of a task or process. Lastly, we searched for additional synonyms used in malware research.

We used the above group of words to construct two database search queries. To make the search more precise, we decided to search only within the abstracts. We conducted the same search in IEEE and ACM databases, two of the largest research databases in computer science and engineering [4]. Two of the top security conferences, USENIX security, and NDSS, along with RAID, a conference with a historical emphasis on malware analysis, are not included in these databases. To find relevant papers published in these three conferences, we created an additional query for Google Scholar. Due to the limitations of Google Scholar search, we were not able to have an identical search query. However, the search strings were logically checked by multiple authors. All of the papers found through the database search underwent the same inclusion and exclusion criteria applied in the manual search described earlier.

2.3 Search Evaluation

To assess the quality of our search results, we conducted tests using known papers found in existing surveys [1, 10]. The first test was on 14 papers found in Table 3 from Afianian et al. [1]. While their table includes 17 papers, only 14 can be found in the databases we searched. Among these 14 papers, our initial database search was able to find 10, achieving a 71.4% retrieval rate. To improve the search, we identified reasons for the missing papers and added synonyms to the query search (ex: "avoid detection", "running", "transparently", "stealthily"). This change increased our retrieval rate to 85.7%, which is above the suggested range of 70%-80% by Kitchenham et al. [37], and added 76 papers to our total database results. To further assess the quality of our search, we conducted another test using 17 papers found in Table 4 of Bulazel et al. [10] (excluding Android and Web papers). Our database search successfully identified 14 of the 17 papers, achieving an 82.4% retrieval rate.

We narrowed our scope to papers published in class A or A* conferences (based on CORE ranking) to help ensure the quality of the papers in this study, as done in prior work [5, 55]. These conferences are known for their rigorous reviewing process. We recognize that limiting our mapping to these conferences may miss relevant papers. However, like other systematic mappings, we do not claim completeness [36, 61] though we strive to ensure quality through our evaluation.

2.4 Data Extraction and Classification

Based on our systematic mapping research questions (MQ1 and MQ2), we developed a standardized data extraction form to ensure consistency in the information gathered from each paper. This form asks for the type of evasion technique that each paper helps address, the methodology used, and the main research question being answered. The first two authors applied this form in a similar manner to reduce bias. Provided

[3] See M-Trends 2023

[4] We observed that the ACM database search result often included research papers that do not match the provided search query, resulting in a high number of excluded papers.

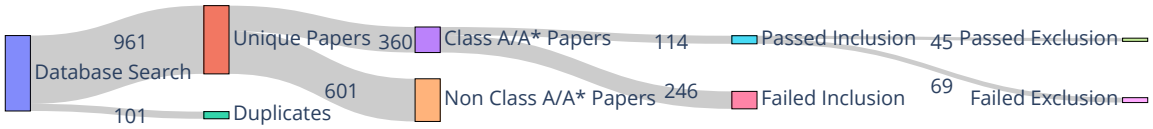

Figure 1: Papers Selection Process

Database	Dynamic Analysis Evasion Papers	Static Analysis Evasion Papers	Total Papers
IEEE	124	195	319
ACM	335	242	577
Google Scholar	86	80	166
Total Papers	545	517	**1062**

Table 1: Database Search Results

with the extracted data, the two authors were able to identify research papers that aimed to answer similar research questions and also address similar evasion techniques. Within the research papers with similar evasion techniques, we further analyzed each research paper to find commonalities among their methodologies. Through this process, we identified several papers that addressed both anti-sandbox and anti-debugging techniques so we categorized these papers into both.

3 Systematic Mapping Results

In this section, we present the results of the systematic mapping. As shown in Figure 1, with the database search, we found 1062 papers. The number of papers identified by each database query search can also be seen in Table 1. After removing duplicates and refining our search to include only class A or A*, we were left with a set of 360 papers. These 360 were then subjected to our inclusion criteria, defined in §2.2, resulting in the selection of 114 papers. Finally, after a thorough assessment of the full text, we applied our exclusion criteria and identified that the majority of the 114 papers do not directly help counter any evasion technique, which resulted in a final set of 45 papers.

Based on the number of papers that address each type of evasion technique, we find that obfuscation and anti-sandbox were the most researched evasion techniques and anti-disassembly was the least. Although we acknowledge that paper counts do not provide a complete explanation for the observed patterns, they serve as a practical and widely accepted metric for identifying trends and patterns in existing literature [2, 11, 59, 62]. Below we explain each of the four categories discovered in our mapping and provide a description of the different methodologies proposed in research to counter them. The categorized papers can be found in Table 2.

Obfuscation. Obfuscation is an evasive technique that modifies the original malware code to obscure the understanding of its functionality. Through our systematic mapping, we identified 19 papers that proposed countermeasures to this type of evasion. The majority of papers used some form of dynamic analysis to overcome the obfuscation [12, 13, 17, 49, 84, 91]. One paper by Coogan et al. [17] deobfuscates virtualized malware by identifying the system calls made when the malware executes and extracting a subtrace containing only the code related to those calls and then approximating the original code with this information. Another frequently used methodology is symbolic execution, which builds expressions containing different inputs and uses a SAT solver to find values that satisfy the expressions [7, 52, 82, 85]. One paper proposed backward-bounded dynamic symbolic execution [7], which leverages symbolic execution to answer infeasibility questions that are frequent with obfuscated code. Researchers have also applied static analysis to overcome obfuscation [46, 66]. Lu et al. [46] proposed a method to remove return-oriented programming (ROP) from malware to enable standard analysis tools to work properly on such malware.

Some papers show that a combination of static and dynamic analysis can also be effective for overcoming obfuscation [14, 63, 64]. PolyUnpack [64] is one such paper that does static analysis to build a static code model, then, during dynamic analysis, compares executed code to this model to identify when unseen code is found to unpack a sample. Finally, there are miscellaneous methodologies that are part of our mapping including program synthesis [8, 29], artificial intelligence-based search [51], and other [44].

Anti-sandbox. Anti-sandbox is an evasive technique used by malware to detect whether its execution is monitored in a controlled environment such as a sandbox. A few of the most common examples of anti-sandbox are system checks, user activity, and delay execution. Through our systematic mapping results, we identified 19 papers that propose countermeasures to this type of evasion. There are three main methodologies that these papers follow. The first one is hypervisor-based analysis [20, 43, 54]. This approach involves the use of virtualization to create isolated environments. To further improve hypervisor-based analysis, these papers focus on creating more transparent hypervisors. For example, Ether [20] proposed a novel and more transparent application of hardware virtualization extensions where the analysis engine resides completely outside the target OS environment. The second methodology is forced execution-based analysis [38, 58, 89]. Forced execution, similar to other path exploration approaches, forces malware to execute through many dif-

Evasion Types	Evasive Tactics	Methodologies	Research Papers
Static	Obfuscation	Dynamic Analysis	2007: [49], 2011: [17], 2013: [91], 2015: [84], 2018: [13], 2023: [12]
		Static Analysis	2011: [46], 2021: [66]
		Dynamic & Static Analysis	2006: [64], 2010: [63], 2021: [14]
		Symbolic Execution	2015: [52] [85], 2017: [7], 2018: [82]
		Other	2010: [29], 2017: [8], 2021: [51], 2022: [44]
	Anti-Disassembly	Dynamic & Static Analysis	2015: [9]
		Static Analysis	2004: [39]
Dynamic	Anti-Sandbox	Bare Metal-Based Analysis	2013: [93]
		Hypervisor-Based Analysis	2008: [20], 2009: [54], 2014: [43], 2015: [90]
		Introspection-Based Analysis	2016: [68], 2021: [71]
		Forced Execution-Based Analysis	2011: [38], 2014: [58], 2020: [89]
		Other	2006: [75], 2010 [16] 2011: [69], 2012: [87], 2013: [34], 2014: [83]
	Anti-Debugging	Hypervisor-Based Analysis	2013: [19], 2015: [90], 2022: [33]
		Bare Metal-Based Analysis	2015: [92]
		Introspection-Based Analysis	2016: [42] [68]
		Instrumentation-Based Analysis	2021: [26]

Table 2: Categorization of Evasion Countermeasures Research Identified Through the Systematic Mapping

ferent paths to collect additional information regarding the malware's behavior. X-force [58] specifically explores different paths without requiring specific inputs or environmental setups. They achieve this by forcing specific instructions, such as predicates and jump table accesses, to have predefined values. A third somewhat less used methodology by research is introspection-based analysis [68, 71]. Introspection refers to software's ability to examine its internal state during execution. Su et al. [71] introduces a novel Out-of-VM introspection technique called Catcher that traces malicious behavior without altering the target environment through the use of CPU cache. Finally, we found six other research papers [16, 34, 69, 75, 83, 87], each with their own distinct methodology including bare-metal-based analysis [93], static analysis [16], taint analysis [34], and multi-system execution [83].

Anti-debugging. Anti-debugging evasion techniques try to detect and prevent analysis of their code during execution. Unlike sandbox evasion techniques, anti-debugging techniques are less concerned with the execution environment and more focused on preventing the analysis of their code. However, despite these differences, they do share similar methods for countering both evasion techniques. Our systematic mapping identified 7 papers that propose countermeasures for this type of evasion. The two most common methodologies we identified were hypervisor-based analysis [19, 33] and introspection-based analysis [42, 68]. The most recent hypervisor-based analysis research paper found in our systematic mapping introduced HyperDbg [33], a specialized hypervisor-assisted debugger that relies on hardware capabilities like Intel-VT to achieve more transparency in their analysis. Conversely, LO-PHI [68] is an example of how introspection-based analysis can be used to help counter anti-debugging techniques. LO-PHI, introduces physical hardware sensors capable of capturing memory and disk activity during execution, which can be used for analyzing evasive malware samples.

The two other methodologies in our mapping were bare metal-based analysis and instrumentation-based analysis. By executing the malware on bare metal and leveraging System Management Mode, MALT [92] enhances the transparency of the execution environment and minimizes the artifacts exposed to malware, which aids in the debugging of evasive malware. In contrast, Hong et al. [26] provide transparency for native read, write, or access to the target through a novel approach called Execution Flow Instrumentation (EFI). EFI allows a user-space program to instrument the execution flow of malicious threads across user and kernel space which can help address existing instrumentation limitations in the analysis of malware with anti-debugging techniques.

Anti-disassembly. Anti-disassembly evasion techniques format malware code in such a way that the disassembler incorrectly interprets the bytes and produces assembly code with errors. This causes problems for analysts because it prevents them from performing accurate static analysis of the malware, which is a critical component of some analysts' processes. The first work that addresses this form of evasion is by Kruegel et al. [39]. This paper primarily uses static analysis to perform a modified recursive and statistical disassembly to correctly disassemble malware that is affected by obfuscation. Bonfante et al. [9] focus on disassembling malware with self-modifying code and overlapping instructions. Their solu-

tion uses both static and dynamic analysis by first executing the malware and taking memory snapshots at different points to capture different waves of code. For each wave, the tool disassembled the code using the dynamic trace as a guide.

4 User Study Methodology

Through our systematic mapping, we were able to identify malware evasion countermeasures and the degree to which they have been explored in research. However, to assess whether these countermeasures meet practitioner needs for analyzing evasive malware, we conducted the first user study on malware evasion to determine which evasion techniques still pose challenges for analysts in practice through 24 semi-structured interviews with malware analysis experts. While an observational study was considered for our methodology, they are less common in related literature because their time-consuming nature can deter participation, and the participants' behavior may be altered during observation. For these reasons, along with NSF's guidance on qualitative methods [56], we chose to conduct semi-structured interviews for our study. Additionally, exploratory qualitative research such as semi-structured interviews offered us the flexibility to gain valuable insights into the participants' decision-making process and their challenges based on their first-hand experiences, while still providing a framework to ensure key topics were covered. Our Institutional Review Board (IRB) approved this study and participants signed a consent form before taking part in the study. To ensure the confidentiality of the participants' information, we sent a draft to all the participants prior to submission and provided an opportunity for them to review and request changes.

4.1 Recruitment

To recruit an expert group of malware analysts for this study, we utilized five different sources. We first reached out to a known Slack channel for malware analysis research and sent a description of our study to a security organization mailing list. Additionally, we reached out to personal contacts who are in the field of malware analysis. Finally, we posted a description of the study on Twitter and LinkedIn, which are not restricted to malware analysts. We specifically selected Twitter (now X) given its popularity within the security community. The description in the message explained that we were looking for malware analysts to participate in a research study in order to understand the current state of practice of evasive malware analysis. We reached out to all five sources listed above in November 2022, and included a link to the initial questionnaire for this study.

Participant Screening. Given the wide range of recruitment sources, we had an initial participant screening phase. To verify that the participants have experience with evasive malware, we had prospective participants fill out a questionnaire with 22 questions hosted on Microsoft Forms. We estimated this questionnaire would take 10 minutes to complete. The first set of questions was about their background information and analysis objectives. The majority of the questions focused on their experience analyzing evasive malware. The last subset of questions contained optional, demographic questions. After interviewing our first participant and understanding the significance of anti-disassembly techniques, we decided to add the question "Based on your experience, which of the following categories of anti-analysis is the most challenging?" This decision was made after meticulously considering the number of responses received and determining a way to ask this question to the participants who had already submitted the questionnaire. Given that we only received 7 responses at the time, we decided to modify the questionnaire for future responses and asked this question during the interview to the participants who had previously submitted the questionnaire. The complete questionnaire is provided in Appendix A.

We received a total of 109 responses to the questionnaire. We analyzed each response by determining whether the answers made sense in the context. Additionally, we attempted to verify the identity of those who responded by looking at the requested LinkedIn profiles. 75 responses were discarded due to random responses to questions that did not indicate the responder had malware analysis skills. Additionally, we were not able to confirm the identity of 5 responses through LinkedIn, so we reached out to them and requested additional information that could help us confirm their identity. Unfortunately, they did not respond, so we were forced to exclude them. We invited the remaining 27 respondents to participate in the interview process and 24 scheduled an interview.

4.2 Interview Protocol

For each participant, we conducted an hour-long, semi-structured interview via online video call. For consistency, all interviews were conducted by the same researcher from November 2022 through January 2023. The interview was broken down into three sections, described below, totaling 29 questions, as seen in Appendix B.

Identifying and Analyzing Evasive Malware. The interview began with understanding how participants identify that a malware sample is evasive and how their analysis process differs when analyzing an evasive sample. Additionally, we asked the participants to explain the main challenges they encounter when analyzing evasive malware samples.

Techniques Used for Handling Evasive Malware. The second section of questions focuses on understanding how malware analysis practitioners handle different evasion techniques. We asked them to walk us through examples of how they handle different evasive techniques such as sandbox evasion, and obfuscation. Additionally, we asked the participants what are the most time-consuming and challenging evasive techniques they encounter, and how they handle them.

Use of Existing Tools for Malware Analysis. The last part of the interview questions is about the analysis tools that the participants use during their analysis process. We asked them what tools they use and which are the most helpful. We also wanted to know what aspects of the workflow would benefit from a new automated system and what improvements they want to see. The purpose of these questions was to understand challenges that need to be addressed.

To ensure that the interview questions were complete, we conducted 3 pilot interviews with graduate students who have malware analysis experience and incorporated their feedback.

4.3 Data Collection and Analysis

All the interviews were audio recorded and automatically transcribed with the built-in video call software. To make sure the transcriptions were accurate, the two first authors manually reviewed and corrected transcription errors. The same researchers then coded the interviews using an iterative open coding methodology [70]. First, they independently coded the first 4 interviews and then agreed on an initial set of codes. These codes were then used by each researcher to recode the same 4 interviews. After this process, we compared the codings between the researchers and eliminated codes that were either redundant or too specific and developed the final codebook composed of 80 codes, as seen in Appendix C. With this final codebook, the authors coded the remaining 20 interviews. The Krippendorff's alpha intercoder reliability score was 0.95 [41], indicating very high consistency. The codes from the final codebook are used to identify patterns and produce our results in §6 and §7.

To determine the adequate amount of participants required for our study, we calculated the point at which we reached saturation. Saturation occurs when no new novel themes are found with additional interviews. To compute saturation, we followed Guest et al. [23] by looking at the interviews that were coded after the final codebook was developed and determining how many interviews it took for all the codes to appear. Additionally, we confirmed that no novel themes emerged. Following this procedure, we reached saturation after the 9th interview. To further validate our findings, when we shared a draft of the paper with the participants, we received several responses stating that they enjoyed reading how their personal experiences aligned with our findings.

5 User Study Participants

We interviewed a group of 24 malware analysts. Most participants have more than 7 years of experience with malware analysis and thus can be viewed as expert analysts. By interviewing participants with many years of experience working at well-established security groups in companies such as Mandiant (now Google), GE, Proofpoint, and Cisco, we were able to identify the most challenging evasive techniques and obtain a broad understanding of how evasive malware is handled in practice. By far, the most common degrees completed by the participants were computer science and electrical engineering. Some participants also specialized in related fields such as information technology or computer networks. Finally, 10 participants stated that they acquired their malware analysis skills through a mentor who played a crucial role in shaping their current analysis workflow. A detailed table of our participants' backgrounds can be found in Table 3 in Appendix D.

6 Malware Analysts' Perspective

In this section, we present malware analysts' definition of evasion and their most challenging evasion techniques.

6.1 Definition of Evasion

To gain a better understanding of what practitioners consider evasive malware, we asked each participant to define evasive malware. The majority of the participants consider evasion to be any technique that affects either their static or dynamic analysis process. As P10 said, an evasive sample "has code, which has the primary goal, intended or not, of disrupting analysis. There are a couple of different forms of analysis, dynamic and static, and each of them has [evasion techniques] to make analysis harder." Static analysis evasion techniques are "designed to make static analysis difficult, such as through obfuscation of code and/or data, or other techniques that incur additional steps to deobfuscate it," as P20 states. Another static technique is anti-disassembly, which causes the disassembler to recover incorrect instructions from the binary. To evade dynamic analysis, malware authors "either bypass the instrumentation itself, detect the environment, or logically not expose behaviors. [This can be done through] delayed execution [or by] requiring parameters," as P7 expressed. Dynamic analysis evasion can affect the proper execution of the malware sample in a debugger, sandbox, or any other controlled environment analysts may use to analyze the sample.

6.2 Most Challenging Evasive Techniques

One of the main objectives of this study is to determine the most challenging evasion techniques malware analysts encounter. To answer this question (RQ2), in the questionnaire, we asked what is the most challenging anti-analysis technique they routinely need to handle. The provided options were anti-disassembly, anti-debugging, sandbox evasion, and others. We further refine the option anti-disassembly into 2 categories; obfuscation, and anti-disassembly based on participants' explanations during the interview. This change also facilitates the comparison between malware analysts' challenges and research countermeasures discussed later in §8.

According to the responses provided by the study participants obfuscation was identified as the most challenging

evasion technique analysts routinely have to handle by 9 participants. As P1 explained, "I think in terms of what is probably the biggest problem on the team, just across all malware right now, it's control flow obfuscated malware. That obfuscation can show up in any kind of malware. JavaScript, PowerShell, windows PEs, you name it. It'll be everywhere." Control flow obfuscated malware tries to hide the actual flow of instructions, which makes it hard for analysts to understand the malware's logic. The second most challenging evasion technique was anti-disassembly, which was mentioned by 6 participants. Anti-disassembly techniques affect static analysis by hindering the proper function of the disassembler. P11 explains that "it tends to be very hard to automate your way out of. It's hard to make a generic anti-anti-disassembly tool."

3 participants expressed anti-debugging as being the most challenging. Anti-debugging techniques include checking for the presence of a debugger or altering the code to prevent reverse engineers from stepping through the code while running it in a debugger. As P20 explains, this evasive technique is challenging because "if it is designed in a way that I can't even follow the code execution [...] that makes it really difficult to figure out which blocks I should narrow in on for doing static analysis work, and it makes it really difficult to create detection signatures." Furthermore, only 2 participants mentioned sandbox evasion to be the most challenging. The reason why sandbox evasion is not considered much of a challenge is "largely because the anti-sandbox stuff I could [...] just run out on a real system, and that real system is still instrumented with a lot of the same tools." as P20 stated.

7 Workflows For Handling Evasive Tactics

To inform future research on ways to address the challenges posed by evasion techniques, we studied how expert malware analysts counter evasion techniques to further analyze the malware's behavior. This information gave us an opportunity for an in-depth look into the processes that each analyst follows. It was not surprising to see that each participant employs a somewhat different workflow, as it can be argued that the process of malware analysis involves a certain degree of creativity, similar to an art form. Despite the creative and varied nature of this process, we were able to distill commonalities among participants' workflows and generated 3 distinct workflows malware analysts follow to handle evasive malware.

7.1 How Malware Experts Handle Dynamic Analysis Evasion

To handle dynamic analysis evasion, our participants use one of two workflows, alter the dynamic analysis execution, or resort to static analysis.

Debugging Workflow for Forced Execution. *Workflow 1*,

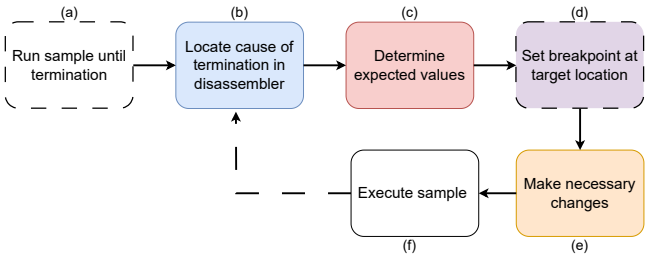

Figure 2: Workflow 1: Forced Execution; P1-P4, P6, P9-P12, P14, P17, P18*, P20* P21, P24*. The asterisk symbol highlights participants that also patch the malware

the first workflow to handle dynamic analysis evasion, shown in Figure 2, is used by 15 of the participants. It is important to note that any line or task with a dotted line is optional. This workflow is used by analysts who either make changes in real-time in the debugger to force the malware sample to execute or patch the malware to create a new version of the malware without evasive checks. To begin, participants execute the malware sample until termination (step a). This can be done in a debugger or a sandbox. Once the sample terminates, the analyst uses the end of the execution trace to locate the code responsible for the evasive check in the disassembly (step b). As P1 stated, "if I'm looking at a sample and I look at the trace and notice that it's stopped there, I can pull it up in IDA, and go right to where it happens." Alternatively, other participants begin this workflow in step b by locating the evasive check without executing it in the debugger. As P4 explained, "the most important parts to understand the [malware's] logic are the conditional branches because that's where the malware is checking, is this true? [...] Is it greater or less than?." After locating the dynamic evasion check, the analysts utilize static analysis to better understand the malware's functionality and determine why the sample stopped executing (step c). This step may also involve identifying the necessary values required for the malware to continue executing.

After gaining a better understanding of the malware's functionality and determining the expected values, analysts either move to step d or step e. The majority of the participants stated a preference towards step d, where they set a breakpoint in the debugger right before the function containing the dynamic evasion technique (step d). Once they execute the sample and reach the breakpoint, the analysts change the value of registers, alter portions of dynamic memory, or flip appropriate bits of conditional jumps to force the malware to continue executing (step e). P17 explained his process as "sometimes it's just as easy as, I change this from 0 to 1, all of a sudden I get a whole new branch. Sometimes it's more complicated [...] and then I have to go into the debugger, change a register or a portion in dynamic memory to read something else, to force the program to do what I want it to do." After the dynamic evasive check is handled, the sample will continue executing until it reaches the next point of termination (step f), at which point, the analyst may decide to repeat this workflow

until the analyst is able to handle additional evasive checks and complete the analysis.

Debugging workflow to execute target functions. One major disadvantage of *Workflow 1* is that analysts have to repeatedly apply it for every evasive check until they reach their goal, which could be tedious, time-consuming, and worse, does not scale. As P2 explains, "it might not always be feasible to get the full code to run because maybe a sample does multiple checks or it's not as straightforward as defeating a check once and letting it execute." To ease this burden, 5 participants prefer to do a more targeted analysis following *Workflow 2* shown in Figure 3. Although the first three steps of this workflow are similar to *Workflow 1*, one main difference is that analysts can choose the point where they begin executing the sample (step d). This point can be after the execution of dynamic analysis evasion techniques in order to avoid handling them. As P1 said, "then I'll just pick random spots after it to start execution again." Analysts may also decide to execute parts of the malware sample that they may find challenging to understand statically. P22 provides an example of when they have used *Workflow 2*, "sometimes it's hard to wrap your mind around an algorithm you're seeing in static analysis. A big loop or some sort of mathematical transform. [...] So you can identify it statically and then go run it." Because the analyst skipped an initial part of the execution, they may have to manually configure the memory and registers for the malware to run properly (step e). This could involve, "thinking about what is being passed to a function and you might have to sort of fabricate parameters to be passed to that code.", as P2 explained. Once the sample is properly configured, the analyst can run the target function and obtain the return value (step f).

Alter Dynamic Analysis Execution. Based on *Workflow 1* and *Workflow 2*, it is evident that the primary tool employed by malware analysts for countering dynamic analysis evasion is a debugger. As reported by 22 participants, the inclination towards using a debugger may be attributed to the limitations of dynamic analysis systems, which fail to capture the entirety of evasive malware's behavior. To mitigate this limitation and handle dynamic analysis evasion, 16 participants either execute the sample in a different system or make alterations to their dynamic analysis system of choice. In such cases, 9 participants choose to run the sample in either commercial sandboxes, internal sandboxes, or a bare metal system, which are more resilient against dynamic analysis evasion. Some of the commercial sandboxes that the participants referred to were VMray, Any.run, Hybrid Analysis, FLARE Sandbox, and Joe Sandbox. Participants stated that these sandboxes incorporate many anti-evasion techniques to handle the most known dynamic analysis evasive techniques. Some participants have access to internal sandboxes that are also able to handle dynamic analysis evasive tactics. As P17 explained, "anytime we come across something like that, like a timing check or a new technique that's trying to look at the environment, we try to build that into the sandbox so that next time

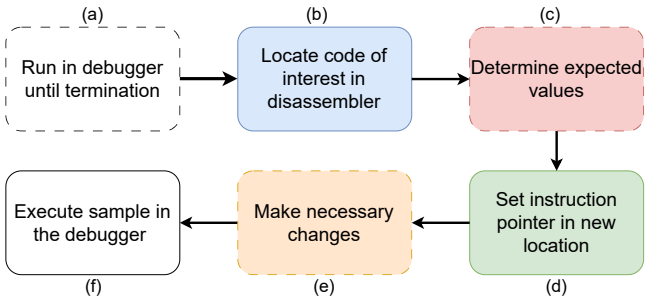

Figure 3: Workflow 2: Targeted Execution; P1, P2, P6, P9, P22

the analyst spins up that sandbox, they don't have to worry about patching over it."

Although these can be effective techniques, not all of the participants have access to such sandboxes. Another option is to manually alter the execution environment to handle the dynamic analysis evasion techniques. As P23 said, "let's try giving a different, fake username just to see [...] in the process of reversing it to try to figure out what it's doing, sometimes you take guesses and think [...] I'm going to try something instead of spending another 2 hours trying to statically reverse it. Let me just try something in 2 minutes with another dynamic analysis run that has some different option." This process generally requires static analysis to determine what changes to make although sometimes analysts will make educated guesses about what changes to make, based on experience. When the analysts are not able to quickly find necessary changes, they often utilize *Workflow 1*, *Workflow 2* or decide to statically analyze the sample.

Analyze the Sample Statically. A few participants who deal with dynamic analysis evasion rely entirely on static analysis as their primary approach. As P11 explained, "since I'm really comfortable with static analysis as opposed to dynamic analysis, I usually just blow through those anti-dynamic analysis measures pretty quickly and I'll just look at it statically and get what I need from it there." This approach is effective for samples that include dynamic analysis evasion techniques but not sophisticated static analysis evasion techniques.

7.2 How Malware Experts Handle Static Analysis Evasion

To handle static analysis evasion, our participants either use workflow 3 or use workflows 1 or 2.

Debugging Workflow for Unpacking. As shown in Figure 4, *Workflow 3* is used by 8 participants primarily to unpack a sample. Additionally, this workflow can also be used to semi-automate the de-obfuscation process. Analysts have expressed that it's easier to execute the malware to de-obfuscate itself than to go through the process statically. For example, P2 said, "I'll just find that in the debugger and let it do all the decoding and then I'll just see what it decoded." The first step in this workflow is to locate the code of interest, which in

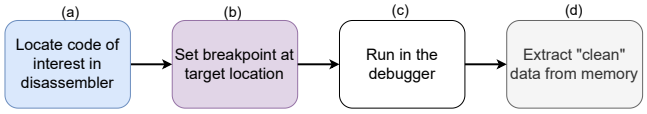

Figure 4: Workflow 3: Unpacking; P2, P4, P5, P6, P7, P20, P22

this case is the function that is responsible for unpacking or deobfuscating the code (step a). This can be done by opening up the binary in the disassembler and "identifying that that's the function that does the string deobfuscation," as P6 stated. After locating this function, the analyst launches a debugger and sets a breakpoint after the identified location (step b), and executes the sample (step c). When the execution reaches the breakpoint, the analyst extracts the deobfuscated data from memory, such as decrypted code, or plain text strings (step d).

Debugging Workflow for Targeted De-obfuscation. *Workflow 3* is an effective way for analysts to handle static analysis evasion when the sample does not include dynamic analysis evasion before the deobfuscation function. However, 10 participants have mentioned that some malware samples implement both static and dynamic analysis evasion. In such cases, the analysts rely on either *Workflow 1* or *Workflow 2*. When utilizing *Workflow 1*, the participants begin executing the sample from the entry point and continue handling each dynamic evasive check until they reach the function that deobfuscates the encrypted data. Once the deobfuscation has been completed, the analyst extracts the data from memory in the same way as the previous workflow.

Although this strategy is effective, participants have expressed that in some cases, malware samples either implement dynamic analysis evasive techniques that are more difficult to handle or include too many dynamic analysis evasive checks, which can be time-consuming. To reduce the analysis time for such samples, participants prefer to use *Workflow 2*, where the code of interest is the function that deobfuscates the target data, such as string decryption. One interesting finding that P2 mentioned is the scenario in which the target function requires parameters to be passed. P2 said "it's getting 4 parameters, what are these parameters? The first one might be where the payload is in the code, the second one might be a key, [...] you might have to do a little bit of work upfront to sort of force it to execute." Following *Workflow 1* and *Workflow 2* allows the participants to extract the information that they want without having to reimplement the samples' decryption algorithm.

8 Comparing Malware Analysts Evasion Challenges and Research Countermeasures

To compare how evasion countermeasures explored by research align with challenges faced in practice, we conducted a comparative analysis between the challenges (§6.2) highlighted by our participants and the solutions found in existing research (§3). As illustrated in Figure 5, although 25% of the participants' stated anti-disassembly as the second most

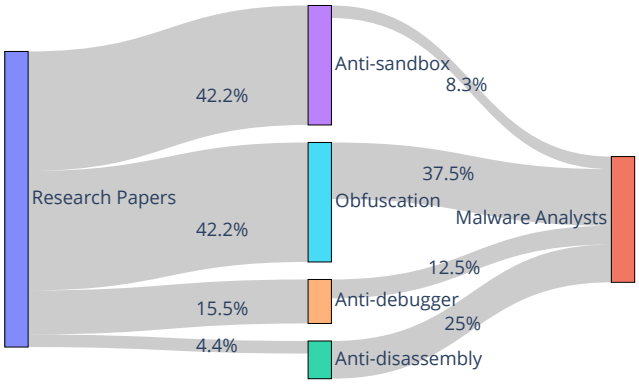

Figure 5: Research on Evasion Technique Countermeasures Vs. Challenging Evasion Techniques for Malware Analysts

challenging evasive technique to handle, based on the volume of identified papers in our systematic mapping, it appears to be less emphasized in research. Our systematic mapping found that this category represented only 4.4% of the research papers, suggesting an opportunity for future research to delve more into this challenging task. Countermeasures for anti-disassembly are especially important now given that participants report a rising trend of malware samples developed in uncommon programming languages, which may lead to inaccuracies in the disassembly and debugging process. As P2 explained, "alternative languages are becoming problematic. So like Golang, Rust, and Delphi are three languages that when you write a program and compile it, it is a lot less straightforward than looking at compiled C." In fact, some participants consider the use of such languages to be a novel evasive technique and participants report that there are not many available tools to deal with this rising problem. Without tools to accurately recover the malware's instructions when it is written in these non-standard languages, analysts' static analysis will be more challenging and less accurate.

As discussed in §6.2, obfuscation is by far the most challenging evasive technique for malware analysts to handle, with 37.5% of the participants stating this observation. At the same time, this evasive technique is by far the one with the most research aimed at developing countermeasures and accounts for 42.2% of the analyzed papers. Based on this observation, we suggest future research directions in §9.

More interestingly, sandbox evasion has the same volume of research papers (42.2%) as obfuscation but is only considered to be a significant challenge by 8.3% of the participants due to their experience with malware analysis, which allows them to circumvent sandbox evasion through static analysis or a debugger, as mentioned in §7. It is worth noting that research has made a lot of strides in creating more resilient and stealthy sandboxes to help handle many techniques used for fingerprinting or environment detection [20, 35, 75, 81, 87]. Despite analysts' ability to handle sandbox evasion techniques, most participants expressed a desire for additional

capabilities. For example, P7 stated that "pure dynamic analysis is very effective for the first stage [...] However, in the second, third to fourth stages, you at least need to somehow convince the attacker to send you them. It might happen not in 4 min. It might happen in 4 days." In such cases, the sandbox may not be able to trigger the behavior that executes each stage of the malware. Without all the stages, analysts are not able to complete their analysis.

Lastly, anti-debugging techniques were identified to be the most challenging evasion to handle by 12.5% of our study participants and account for 15.5% of the papers found through our systematic mapping. Based on the participants' statements, analysts are not in need of additional tools to handle anti-debugging because available tools such as Scyllahide that can be used to handle the majority of these techniques.

9 Discussion

Through our comparative analysis in §8, we are able to obtain a better understanding of the impact of research on the state of practice of evasive malware analysis. Specifically, we identified discrepancies between research and practice, finding instances where challenging techniques lack sufficient research attention and others where significant research exists despite persistent analyst difficulties. To help mitigate the identified discrepancies between research and practice, we discuss future research directions that could help analysts address the challenges they still face when analyzing evasive malware. Additionally, we underscore the need to investigate barriers that impede the transition of research into practice.

Analysis of Malware with Anti-disassembly Techniques. Our results show that anti-disassembly is perceived as a major challenge by many participants, yet in our systematic mapping we found a noticeable gap in research focused on countering evasive malware that implements these techniques. While we acknowledge the difficulties in addressing the challenges of anti-disassembly, our observations reveal a prevalent reliance on disassemblers among malware analysts in practice. Consequently, it is imperative for future research to develop techniques specifically tailored for analyzing malware with anti-disassembly capabilities. More specifically, study participants highlighted a need for ways to counter such evasive malware written in less common programming languages such as Golang, as mentioned in §8. Effective methodologies that counter anti-disassembly can significantly reduce the amount of effort required by analysts to locate the cause of termination (step b in *Workflow 1*), locate the code of interest (step b in *Workflow 2*, step a in *Workflow 3*), and understand the malware's logic when it is written in alternative languages or has evasive techniques that affect the disassembly process.

Evaluation of Existing Research Solutions in Practice (Deobfuscation Tools). Obfuscation is another serious challenge that was mentioned by our study participants. However, considering that previous research has developed many automated systems to deobfuscate malware, as shown in §3, it raises the question of why obfuscation remains a major challenge in analyzing evasive malware in practice. Future research should focus on identifying and addressing any potential barriers that impede the transfer of this research into practice.

Malware Analysis Research with Analysts in Mind. In our analysis, we identified the need for sandboxes tailored to meet malware analysts' specific needs. Prior work on sandboxes focuses on scalability and tries to optimize execution time [40]. As expert analysts focus on more sophisticated malware that employs evasive techniques, they require a more granular report with a longer execution time. As P17 said, "I would rather wait a longer time for a tool to go through and really explore an executable, and automatically go through all the different branches that could be taken." One promising direction for meeting this requirement is symbolic execution. Contrary to forced execution solutions [38, 58, 89], symbolic execution maintains a valid execution state of the malware at all times and is less likely to miss unexpected execution flows that could reveal the malware's malicious behavior. This approach is further motivated by 4 participants who mentioned they use angr [67], a popular open-source binary analysis framework that applies symbolic execution. Unfortunately, deploying this tool in analysts' workflow is difficult because it's not targeted for malware analysis. As P10 said, "angr has very limited applications in malware analysis, and things like a debugger or emulation will work a lot better for me".

To design tools that better meet malware analysts' needs and that more easily integrate with analysts' current analysis process, researchers could follow *Workflow 1* as a guide. A tool could begin by executing the malware sample until it terminates (step a), then it could automatically locate the last conditional branch that was executed before termination through static analysis (step b). The next step in the analysis workflow could be to determine what inputs the malware requires to satisfy the conditional branch (step c). This step is currently a demanding manual process that could benefit from the implementation of symbolic execution. By providing the symbolic execution engine with the conditional branch as a target, it could come up with constraints for the path that did not terminate and generate new inputs that allow the malware to continue executing. Once the inputs are generated, they can either be provided to the malware analyst or set automatically in the dynamic analysis environment to continue executing the malware sample. The implementation of such systems could help analysts observe more of the malware's behavior in a shorter amount of time. Additionally, these systems could also help analyze malware samples that implement both static and dynamic analysis evasion, as participants mentioned the use of Workflow 1 as a way of analyzing these malware samples in §7.2. Ideas along these lines appear in the work of Chipounov et al. [15] but they are applied in the context of reverse engineering and bug finding. Based on our findings, we

believe that these recommendations can help narrow the gap between research advances and practitioner needs, thereby enhancing the overall impact of research in the field.

10 Limitations

Exploratory qualitative research practices have well-known limitations. The first limitation is the lack of complete recall of participants' analysis process [25]. To mitigate this limitation, we asked participants to state as much as they could recall and only after they were done, move to the next question. This is a best practice used to ensure recall [27]. The second limitation is that participants may modify their answers to appear more experienced in the field. We aimed to mitigate this limitation by confirming their identity and years of experience through the participant screening described in §4.1. Given the exploratory nature of this study, the number that we provide in each finding is the number of participants that explicitly stated a concept. However, we acknowledge that this number may be higher as some participants may have failed to state the concept.

As was mentioned previously in §2, we scoped our systematic mapping results to papers published at tier A or tier A* conferences which may exclude some relevant papers. However, in line with the nature of systematic mappings, we do not claim completeness [36, 61]. Instead, we focus on ensuring the quality of our mapping through a rigorous evaluation. It is also important to recognize that although paper counts alone do not offer a comprehensive explanation for the observed patterns, they serve as a practical and widely accepted metric for identifying patterns within existing literature [2, 11, 59].

11 Related Work

To our knowledge, only four human-centered studies have been conducted to understand the cognitive process of software reverse engineering [4, 47, 77, 88]. Of those, two are focused on malware analysis [4, 88]. The first user study with reverse engineers as participants was conducted by Votipka et al. [77] in 2020. This study used semi-structured observational interviews to gain an understanding of the process of reverse engineering. The authors were able to develop workflows that represent the necessary process reverse engineers follow and suggest guidelines for designing future reverse engineering tools. This early research has informed subsequent studies that further explore this area [47], and investigate other related fields such as malware analysis [88]. In 2021, Yong et al. [88] conducted a user study specifically to understand the objectives and workflows of malware analysts in practice. In this study, they proposed a taxonomy of malware analysts and identified workflows employed by the participants. In contrast to [88], we conduct a systematic mapping of countermeasures for malware evasion techniques, which allows us to understand the extent to which malware evasion techniques have been researched. Additionally, we perform a user study to identify the specific evasion techniques that remain challenging for malware analysts in practice. More importantly, our work identifies potential gaps by comparing evasion techniques that remain challenging for practitioners with countermeasures explored in the research literature, which is not addressed in [88]. Thus, other than a similar user study methodology, there is minimal overlap between our research.

Montovani et al. [47] aim to understand the mental model and strategies adopted by reverse engineers to solve static RE tasks. Unlike previous studies using interviews, Montovani et al. designed a web-based platform similar to traditional interactive disassemblers, which allowed a fine-grained observation of the participants' actions. Their findings revealed expert REs visit fewer basic blocks than beginner REs, and identified strategies strongly correlated with experience level.

Recent work by Aonzo et al. [4] compares the procedures followed by humans and machines to classify unknown programs as benign or malicious, aiming to understand how data from malware analysis reports is used to reach a decision. They accomplish this by designing an online game that requests participants to classify suspicious files based on their sandbox reports. During this activity, features required by the analysts are observed. These features are then compared to two Machine Learning (ML)-based malware classification models. The key finding of this study is that the ML algorithms performed less effectively and do not use the same features as the malware analysts. ML focuses on static features, while humans rely more on dynamic features.

Although it is only recently that researchers have started to study cognitive procedures in malware analysis and reverse engineering, it is important to acknowledge the considerable amount of usable security research. Its predominant focus has been on secure software development [6, 22, 57, 72, 76] and vulnerability detection [3, 24, 78]. Prior work has also tested the usability of tools used by reverse engineers [50, 86].

12 Conclusion

In this study, we focus on malware evasion techniques and their countermeasures. We conduct a systematic mapping of research in evasion countermeasures. Additionally, we conducted a user study with malware experts to determine the most challenging evasive techniques that analysts encounter in practice. This allowed us to identify three distinct workflows that capture the process that analysts use to handle malware evasion. Lastly, we performed a comparative analysis of solutions explored by research and challenges encountered in practice. Such analysis can inform future research directions that can help analysts handle challenging evasive techniques. It can also help understand potential barriers that may be hindering the transition of research into practice.

References

[1] Amir Afianian, Salman Niksefat, Babak Sadeghiyan, and David Baptiste. “Malware dynamic analysis evasion techniques: A survey”. In: *ACM Computing Surveys (CSUR)* 52.6 (2019), pp. 1–28.

[2] Muhammad Ovais Ahmad, Denis Dennehy, Kieran Conboy, and Markku Oivo. “Kanban in software engineering: A systematic mapping study”. In: *Journal of Systems and Software* 137 (2018), pp. 96–113.

[3] Noura Alomar, Primal Wijesekera, Edward Qiu, and Serge Egelman. “" You’ve got your nice list of bugs, now what?" vulnerability discovery and management processes in the wild”. In: *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*. 2020, pp. 319–339.

[4] Simone Aonzo, Yufei Han, Alessandro Mantovani, and Davide Balzarotti. “Humans vs. Machines in Malware Classification”. In: *32nd USENIX Security Symposium (USENIX Security 23)*. 2023, pp. 1145–1162.

[5] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. “Dos and don’ts of machine learning in computer security”. In: *31st USENIX Security Symposium (USENIX Security 22)*. 2022, pp. 3971–3988.

[6] Hala Assal and Sonia Chiasson. “Security in the Software Development Lifecycle.” In: *Fourteenth symposium on usable privacy and security (SOUPS 2018)*. 2018, pp. 281–296.

[7] Sébastien Bardin, Robin David, and Jean-Yves Marion. “Backward-bounded DSE: targeting infeasibility questions on obfuscated codes”. In: *2017 IEEE Symposium on Security and Privacy*. IEEE. 2017, pp. 633–651.

[8] Tim Blazytko, Moritz Contag, Cornelius Aschermann, and Thorsten Holz. “Syntia: Synthesizing the Semantics of Obfuscated Code.” In: *26th USENIX Security Symposium (USENIX Security 17)*. 2017, pp. 643–659.

[9] Guillaume Bonfante, Jose Fernandez, Jean-Yves Marion, Benjamin Rouxel, Fabrice Sabatier, and Aurélien Thierry. “Codisasm: Medium scale concatic disassembly of self-modifying binaries with overlapping instructions”. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 2015, pp. 745–756.

[10] Alexei Bulazel and Bülent Yener. “A survey on automated dynamic malware analysis evasion and counter-evasion: Pc, mobile, and web”. In: *Proceedings of the 1st Reversing and Offensive-oriented Trends Symposium*. 2017, pp. 1–21.

[11] Anna Cantrell, Elizabeth Croot, Maxine Johnson, Ruth Wong, Duncan Chambers, Susan K Baxter, and Andrew Booth. “Access to primary and community health-care services for people 16 years and over with intellectual disabilities: a mapping and targeted systematic review”. In: (2020).

[12] Binlin Cheng, Erika A Leal, Haotian Zhang, and Jiang Ming. “On the Feasibility of Malware Unpacking via Hardware-assisted Loop Profiling”. In: *32nd USENIX Security Symposium (USENIX Security 23)*. 2023, pp. 7481–7498.

[13] Binlin Cheng, Jiang Ming, Jianmin Fu, Guojun Peng, Ting Chen, Xiaosong Zhang, and Jean-Yves Marion. “Towards paving the way for large-scale windows malware analysis: Generic binary unpacking with orders-of-magnitude performance boost”. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 395–411.

[14] Binlin Cheng, Jiang Ming, Erika A Leal, Haotian Zhang, Jianming Fu, Guojun Peng, and Jean-Yves Marion. “{Obfuscation-Resilient} Executable Payload Extraction From Packed Malware”. In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, pp. 3451–3468.

[15] Vitaly Chipounov, Volodymyr Kuznetsov, and George Candea. “S2E: A platform for in-vivo multi-path analysis of software systems”. In: *Acm Sigplan Notices* 46.3 (2011), pp. 265–278.

[16] Paolo Milani Comparetti, Guido Salvaneschi, Engin Kirda, Clemens Kolbitsch, Christopher Kruegel, and Stefano Zanero. “Identifying dormant functionality in malware programs”. In: *2010 IEEE Symposium on Security and Privacy*. IEEE. 2010, pp. 61–76.

[17] Kevin Coogan, Gen Lu, and Saumya Debray. “Deobfuscation of virtualization-obfuscated software: a semantics-based approach”. In: *Proceedings of the 18th ACM conference on Computer and communications security*. 2011, pp. 275–284.

[18] Mila Dalla Preda, Matias Madou, Koen De Bosschere, and Roberto Giacobazzi. “Opaque predicates detection by abstract interpretation”. In: *Algebraic Methodology and Software Technology: 11th International Conference, AMAST 2006, Kuressaare, Estonia, July 5-8, 2006. Proceedings 11*. Springer. 2006, pp. 81–95.

[19] Zhui Deng, Xiangyu Zhang, and Dongyan Xu. “Spider: Stealthy binary program instrumentation and debugging via hardware virtualization”. In: *Proceedings of the 29th Annual Computer Security Applications Conference*. 2013, pp. 289–298.

[20] Artem Dinaburg, Paul Royal, Monirul Sharif, and Wenke Lee. “Ether: malware analysis via hardware virtualization extensions”. In: *Proceedings of the 15th ACM conference on Computer and communications security*. 2008, pp. 51–62.

[21] Daniele Cono D’Elia, Emilio Coppa, Federico Palmaro, and Lorenzo Cavallaro. “On the dissection of evasive malware”. In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 2750–2765.

[22] Kelsey R Fulton, Daniel Votipka, Desiree Abrokwa, Michelle L Mazurek, Michael Hicks, and James Parker. “Understanding the how and the why: Exploring secure development practices through a course competition”. In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2022, pp. 1141–1155.

[23] Greg Guest, Emily Namey, and Mario Chen. “A simple method to assess and report thematic saturation in qualitative research”. In: *PloS one* 15.5 (2020), e0232076.

[24] Marco Gutfleisch, Jan H Klemmer, Niklas Busch, Yasemin Acar, M Angela Sasse, and Sascha Fahl. “How does usable security (not) end up in software products? results from a qualitative interview study”. In: *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2022, pp. 893–910.

[25] Erik Hollnagel. *Handbook of cognitive task design*. CRC Press, 2003.

[26] Jiaqi Hong and Xuhua Ding. “A novel dynamic analysis infrastructure to instrument untrusted execution flow across user-kernel spaces”. In: *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2021, pp. 1902–1918.

[27] Stacy A Jacob and S Paige Furgerson. “Writing interview protocols and conducting interviews: Tips for students new to the field of qualitative research.” In: *Qualitative Report* 17 (2012), p. 6.

[28] Ashish Jadhav, Deepti Vidyarthi, and M Hemavathy. “Evolution of evasive malwares: A survey”. In: *2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT)*. IEEE. 2016, pp. 641–646.

[29] Susmit Jha, Sumit Gulwani, Sanjit A Seshia, and Ashish Tiwari. “Oracle-guided component-based program synthesis”. In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. 2010, pp. 215–224.

[30] Noah M Johnson, Juan Caballero, Kevin Zhijie Chen, Stephen McCamant, Pongsin Poosankam, Daniel Reynaud, and Dawn Song. “Differential slicing: Identifying causal execution differences for security applications”. In: *2011 IEEE Symposium on Security and Privacy*. IEEE. 2011, pp. 347–362.

[31] Min Gyung Kang, Pongsin Poosankam, and Heng Yin. “Renovo: A hidden code extractor for packed executables”. In: *Proceedings of the 2007 ACM workshop on Recurring malcode*. 2007, pp. 46–53.

[32] Min Gyung Kang, Heng Yin, Steve Hanna, Stephen McCamant, and Dawn Song. “Emulating emulation-resistant malware”. In: *Proceedings of the 1st ACM workshop on Virtual machine security*. 2009, pp. 11–22.

[33] Mohammad Sina Karvandi, MohammadHosein Gholamrezaei, Saleh Khalaj Monfared, Soroush Meghdadizanjani, Behrooz Abbassi, Ali Amini, Reza Mortazavi, Saeid Gorgin, Dara Rahmati, and Michael Schwarz. “HyperDbg: Reinventing Hardware-Assisted Debugging”. In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2022, pp. 1709–1723.

[34] Yuhei Kawakoya, Makoto Iwamura, Eitaro Shioji, and Takeo Hariu. “Api chaser: Anti-analysis resistant malware analyzer”. In: *Research in Attacks, Intrusions, and Defenses: 16th International Symposium, RAID 2013, Rodney Bay, St. Lucia, October 23-25, 2013. Proceedings 16*. Springer. 2013, pp. 123–143.

[35] Dhilung Kirat, Giovanni Vigna, and Christopher Kruegel. “Barecloud: Bare-metal analysis-based evasive malware detection”. In: *23rd USENIX Security Symposium (USENIX Security 14)*. 2014, pp. 287–301.

[36] Barbara A Kitchenham, David Budgen, and O Pearl Brereton. “The value of mapping studies–A participant-observer case study”. In: *14th international conference on evaluation and assessment in software engineering (ease)*. 2010, pp. 1–9.

[37] Barbara Ann Kitchenham, David Budgen, and Pearl Brereton. *Evidence-based software engineering and systematic reviews*. Vol. 4. CRC press, 2015.

[38] Clemens Kolbitsch, Engin Kirda, and Christopher Kruegel. “The power of procrastination: detection and mitigation of execution-stalling malicious code”. In: *Proceedings of the 18th ACM conference on Computer and communications security*. 2011, pp. 285–296.

[39] Christopher Kruegel, William Robertson, Fredrik Valeur, and Giovanni Vigna. “Static disassembly of obfuscated binaries”. In: *USENIX security Symposium*. Vol. 13. 2004, pp. 18–18.

[40] Alexander Küchler, Alessandro Mantovani, Yufei Han, Leyla Bilge, and Davide Balzarotti. “Does Every Second Count? Time-based Evolution of Malware Behavior in Sandboxes.” In: *NDSS*. 2021.

[41] J Richard Landis and Gary G Koch. “An application of hierarchical kappa-type statistics in the assessment of majority agreement among multiple observers”. In: *Biometrics* (1977), pp. 363–374.

[42] Kevin Leach, Chad Spensky, Westley Weimer, and Fengwei Zhang. “Towards transparent introspection”. In: *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. Vol. 1. IEEE. 2016, pp. 248–259.

[43] Tamas K Lengyel, Steve Maresca, Bryan D Payne, George D Webster, Sebastian Vogl, and Aggelos Kiayias. “Scalability, fidelity and stealth in the DRAKVUF dynamic malware analysis system”. In: *Proceedings of the 30th annual computer security applications conference*. 2014, pp. 386–395.

[44] Shijia Li, Chunfu Jia, Pengda Qiu, Qiyuan Chen, Jiang Ming, and Debin Gao. “Chosen-Instruction Attack Against Commercial Code Virtualization Obfuscators”. In: *In Proceedings of the 29th Network and Distributed System Security Symposium*. 2022.

[45] Martina Lindorfer, Clemens Kolbitsch, and Paolo Milani Comparetti. “Detecting environment-sensitive malware”. In: *Recent Advances in Intrusion Detection: 14th International Symposium, RAID 2011, Menlo Park, CA, USA, September 20-21, 2011. Proceedings 14*. Springer. 2011, pp. 338–357.

[46] Kangjie Lu, Dabi Zou, Weiping Wen, and Debin Gao. “deRop: removing return-oriented programming from malware”. In: *Proceedings of the 27th Annual Computer Security Applications Conference*. 2011, pp. 363–372.

[47] Alessandro Mantovani, Simone Aonzo, Yanick Fratantonio, and Davide Balzarotti. “{RE-Mind}: a First Look Inside the Mind of a Reverse Engineer”. In: *31st USENIX Security Symposium (USENIX Security 22)*. 2022, pp. 2727–2745.

[48] Jonathan AP Marpaung, Mangal Sain, and Hoon-Jae Lee. “Survey on malware evasion techniques: State of the art and challenges”. In: *2012 14th International Conference on Advanced Communication Technology (ICACT)*. IEEE. 2012, pp. 744–749.

[49] Lorenzo Martignoni, Mihai Christodorescu, and Somesh Jha. “Omniunpack: Fast, generic, and safe unpacking of malware”. In: *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*. IEEE. 2007, pp. 431–441.

[50] James Mattei, Madeline McLaughlin, Samantha Katcher, and Daniel Votipka. “A Qualitative Evaluation of Reverse Engineering Tool Usability”. In: *Proceedings of the 38th Annual Computer Security Applications Conference*. 2022, pp. 619–631.

[51] Grégoire Menguy, Sébastien Bardin, Richard Bonichon, and Cauim de Souza Lima. “Search-based local black-box deobfuscation: understand, improve and mitigate”. In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2021, pp. 2513–2525.

[52] Jiang Ming, Dongpeng Xu, Li Wang, and Dinghao Wu. “Loop: Logic-oriented opaque predicate detection in obfuscated binary code”. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 2015, pp. 757–768.

[53] Andreas Moser, Christopher Kruegel, and Engin Kirda. “Exploring multiple execution paths for malware analysis”. In: *2007 IEEE Symposium on Security and Privacy (SP’07)*. IEEE. 2007, pp. 231–245.

[54] Anh M Nguyen, Nabil Schear, HeeDong Jung, Apeksha Godiyal, Samuel T King, and Hai D Nguyen. “Mavmm: Lightweight and purpose built vmm for malware analysis”. In: *2009 Annual Computer Security Applications Conference*. IEEE. 2009, pp. 441–450.

[55] Anna-Marie Ortloff, Christian Tiefenau, and Matthew Smith. “{SoK}: I Have the (Developer) Power! Sample Size Estimation for Fisher’s Exact,{Chi-Squared},{McNemar’s}, Wilcoxon {Rank-Sum}, Wilcoxon {Signed-Rank} and t-tests in {Developer-Centered} Usable Security”. In: *Nineteenth Symposium on Usable Privacy and Security (SOUPS 2023)*. 2023, pp. 341–359.

[56] *Overview of Qualitative Methods and Analytic Techniques*. `https://www.nsf.gov/pubs/1997/nsf97153/chap_3.htm`.

[57] Hernan Palombo, Armin Ziaie Tabari, Daniel Lende, Jay Ligatti, and Xinming Ou. “An ethnographic understanding of software (in) security and a co-creation model to improve secure software development”. In: *Proceedings of the Sixteenth Symposium on Usable Privacy and Security*. 2020.

[58] Fei Peng, Zhui Deng, Xiangyu Zhang, Dongyan Xu, Zhiqiang Lin, and Zhendong Su. “X-force: Force-executing binary programs for security applications”. In: *USENIX Security symposium 2014*.

[59] Birgit Penzenstadler, Ankita Raturi, Debra Richardson, Coral Calero, Henning Femmer, and Xavier Franch. “Systematic mapping study on software engineering for sustainability (SE4S)”. In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. 2014, pp. 1–14.

[60] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. “Systematic mapping studies in software engineering”. In: *12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12*. 2008, pp. 1–10.

[61] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. “Guidelines for conducting systematic mapping studies in software engineering: An update”. In: *Information and software technology* 64 (2015), pp. 1–18.

[62] Meisam Ranjbari, Zahra Shams Esfandabadi, Tetiana Shevchenko, Naciba Chassagnon-Haned, Wanxi Peng, Meisam Tabatabaei, and Mortaza Aghbashlo. “Mapping healthcare waste management research: Past evolution, current challenges, and future perspectives towards a circular economy transition”. In: *Journal of hazardous materials* 422 (2022), p. 126724.

[63] Kevin A Roundy and Barton P Miller. “Hybrid analysis and control of malware”. In: *Recent Advances in Intrusion Detection: 13th International Symposium, RAID 2010, Ottawa, Ontario, Canada, September 15-17, 2010. Proceedings 13*. Springer. 2010, pp. 317–338.

[64] Paul Royal, Mitch Halpin, David Dagon, Robert Edmonds, and Wenke Lee. “PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware”. In: *2006 22nd Annual Computer Security Applications Conference (ACSAC’06)*. 2006, pp. 289–300. DOI: 10.1109/ACSAC.2006.38.

[65] Monirul Sharif, Andrea Lanzi, Jonathon Giffin, and Wenke Lee. “Automatic reverse engineering of malware emulators”. In: *2009 30th IEEE Symposium on Security and Privacy*. IEEE. 2009, pp. 94–109.

[66] Junfu Shen and Jiang Ming. “Mba-blast: unveiling and simplifying mixed boolean-arithmetic obfuscation”. In: (2021).

[67] Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Audrey Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. “SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis”. In: *IEEE Symposium on Security and Privacy*. 2016.

[68] Chad Spensky, Hongyi Hu, and Kevin Leach. “LO-PHI: Low-Observable Physical Host Instrumentation for Malware Analysis.” In: *NDSS*. 2016.

[69] Deepa Srinivasan, Zhi Wang, Xuxian Jiang, and Dongyan Xu. “Process out-grafting: an efficient" out-of-vm" approach for fine-grained process execution monitoring”. In: *Proceedings of the 18th ACM conference on Computer and communications security*. 2011, pp. 363–374.

[70] Prachi Srivastava and Nick Hopwood. “A practical iterative framework for qualitative data analysis”. In: *International journal of qualitative methods* 8.1 (2009), pp. 76–84.

[71] Chao Su, Xuhua Ding, and Qingkai Zeng. “Catch you with cache: Out-of-VM introspection to trace malicious executions”. In: *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE. 2021, pp. 326–337.

[72] Tyler W Thomas, Madiha Tabassum, Bill Chu, and Heather Lipford. “Security during application development: An application security expert perspective”. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 2018, pp. 1–12.

[73] Sharath K Udupa, Saumya K Debray, and Matias Madou. “Deobfuscation: Reverse engineering obfuscated code”. In: *12th Working Conference on Reverse Engineering (WCRE’05)*. IEEE. 2005, 10–pp.

[74] Xabier Ugarte-Pedrero, Davide Balzarotti, Igor Santos, and Pablo G Bringas. “Rambo: Run-time packer analysis with multiple branch observation”. In: *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings 13*. Springer. 2016, pp. 186–206.

[75] Amit Vasudevan and Ramesh Yerraballi. “Cobra: Fine-grained malware analysis using stealth localized-executions”. In: *2006 IEEE Symposium on Security and Privacy (S&P’06)*. IEEE. 2006, 15–pp.

[76] Daniel Votipka, Kelsey R Fulton, James Parker, Matthew Hou, Michelle L Mazurek, and Michael Hicks. “Understanding security mistakes developers make: Qualitative analysis from build it, break it, fix it”. In: *29th USENIX Security Symposium (USENIX Security 20)*. 2020, pp. 109–126.

[77] Daniel Votipka, Seth M Rabin, Kristopher Micinski, Jeffrey S Foster, and Michelle M Mazurek. “An observational investigation of reverse engineers’ processes”. In: *Proceedings of the 29th USENIX Conference on Security Symposium*. 2020, pp. 1875–1892.

[78] Daniel Votipka, Rock Stevens, Elissa Redmiles, Jeremy Hu, and Michelle Mazurek. “Hackers vs. testers: A comparison of software vulnerability discovery processes”. In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2018, pp. 374–391.

[79] Jeffrey Wilhelm and Tzi-cker Chiueh. “A forced sampled execution approach to kernel rootkit identification”. In: *Recent Advances in Intrusion Detection: 10th International Symposium, RAID 2007, Gold Goast, Australia, September 5-7, 2007. Proceedings 10*. Springer. 2007, pp. 219–235.

[80] Carsten Willems, Thorsten Holz, and Felix Freiling. "Toward automated dynamic malware analysis using cwsandbox". In: *IEEE Security & Privacy* 5.2 (2007), pp. 32–39.

[81] Carsten Willems, Ralf Hund, Andreas Fobian, Dennis Felsch, Thorsten Holz, and Amit Vasudevan. "Down to the bare metal: Using processor features for binary analysis". In: *Proceedings of the 28th Annual Computer Security Applications Conference*. 2012, pp. 189–198.

[82] Dongpeng Xu, Jiang Ming, Yu Fu, and Dinghao Wu. "VMHunt: A verifiable approach to partially-virtualized binary code simplification". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 442–458.

[83] Zhaoyan Xu, Jialong Zhang, Guofei Gu, and Zhiqiang Lin. "Goldeneye: Efficiently and effectively unveiling malware's targeted environment". In: *Research in Attacks, Intrusions and Defenses: 17th International Symposium, RAID 2014, Gothenburg, Sweden, September 17-19, 2014. Proceedings 17*. Springer. 2014, pp. 22–45.

[84] Babak Yadegari and Saumya Debray. "Symbolic execution of obfuscated code". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 2015, pp. 732–744.

[85] Babak Yadegari, Brian Johannesmeyer, Ben Whitely, and Saumya Debray. "A generic approach to automatic deobfuscation of executable code". In: *2015 IEEE Symposium on Security and Privacy*. IEEE. 2015, pp. 674–691.

[86] Khaled Yakdan, Sergej Dechand, Elmar Gerhards-Padilla, and Matthew Smith. "Helping johnny to analyze malware: A usability-optimized decompiler and malware analysis user study". In: *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2016, pp. 158–177.

[87] Lok-Kwong Yan, Manjukumar Jayachandra, Mu Zhang, and Heng Yin. "V2e: combining hardware virtualization and softwareemulation for transparent and extensible malware analysis". In: *Proceedings of the 8th ACM SIGPLAN/SIGOPS conference on Virtual Execution Environments*. 2012, pp. 227–238.

[88] Miuyin Yong Wong, Matthew Landen, Manos Antonakakis, Douglas M Blough, Elissa M Redmiles, and Mustaque Ahamad. "An inside look into the practice of malware analysis". In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2021, pp. 3053–3069.

[89] Wei You, Zhuo Zhang, Yonghwi Kwon, Yousra Aafer, Fei Peng, Yu Shi, Carson Harmon, and Xiangyu Zhang. "Pmp: Cost-effective forced execution with probabilistic memory pre-planning". In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2020, pp. 1121–1138.

[90] Junyuan Zeng, Yangchun Fu, and Zhiqiang Lin. "Pemu: A pin highly compatible out-of-vm dynamic binary instrumentation framework". In: *Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*. 2015, pp. 147–160.

[91] Junyuan Zeng, Yangchun Fu, Kenneth A Miller, Zhiqiang Lin, Xiangyu Zhang, and Dongyan Xu. "Obfuscation resilient binary code reuse through trace-oriented programming". In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 2013, pp. 487–498.

[92] Fengwei Zhang, Kevin Leach, Angelos Stavrou, Haining Wang, and Kun Sun. "Using hardware features for increased debugging transparency". In: *2015 IEEE Symposium on Security and Privacy*. IEEE. 2015, pp. 55–69.

[93] Fengwei Zhang, Kevin Leach, Kun Sun, and Angelos Stavrou. "Spectre: A dependable introspection framework via system management mode". In: *2013 43rd Annual IEEE/IFIP international conference on dependable systems and networks (DSN)*. IEEE. 2013, pp. 1–12.

[94] Qinghua Zhang and Douglas S Reeves. "Metaaware: Identifying metamorphic malware". In: *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*. IEEE. 2007, pp. 411–420.

A Survey Questionnaire

Background and Experience.

- How many years of experience do you have analyzing malware?
- Can you please describe your job role?
- We would like to get to know more about you, please provide your LinkedIn profile. If you do not have a LinkedIn profile, please describe your work experience and education such as your highest level of education and major (if applicable).
- Which of the following best describes your malware analysis objective? Extract easily obtained string based IOCs such as hashes, domain names and IP addresses from malware samples, Focus on identifying potentially malicious activities exhibited by malware samples using network and host artifacts, Perform malware analysis to identify the strategies and intentions behind threat actor's attack campaigns which is accomplished by understanding the Tactics, Techniques and Procedures (check all that apply).

Malware Evasion.

- How do you define an evasive malware sample?
- How often do you analyze evasive malware?
- How do you determine that a malware sample is evasive?
- Which of the following type of analysis do you tend to rely on more when identifying an evasive malware sample? Dynamic Analysis, Static Analysis, I use both static and dynamic analysis equally, Other.
- Which of the following type of analysis do you tend to rely on more when analyzing evasive malware samples? Dynamic Analysis, Static Analysis, I use both static and dynamic analysis equally, Other.
- Do you consider evasive malware to be challenging to analyze? Yes, No, Other.
- What is the biggest challenge when analyzing evasive malware samples?
- Based on your experience, have evasive tactics become more sophisticated over time? If so, how?
- Based on your experience, is there any correlation between the malware family and the evasive tactics?
- Based on your experience, which of the following category of anti-analysis is the most challenging? Anti-disassembly, Anti-debugging, Sandbox Evasion, Other.
- Based on your experience, what are the most common types of sandbox evasion tactics? Delayed Execution: execute malware after a short period of time to successfully leave the sandbox, Environmental Awareness: verify that it is being executed in a real life environment, System Analysis: look for system characteristics like CPU core count and system reboots, User Interaction: detect user actions like mouse clicks and document scrolling, Data Obfuscation: tricks the sandbox by changing the DNS names or encrypting API calls.
- Does your analysis process differ when you are analyzing an evasive malware sample? If so, how?

B Interview Questions

Identifying and Analyzing Evasive Malware.

- At a high level, what is your workflow when analyzing a malware sample?
- When do you begin to consider the possibility that the sample is evasive?
- How do you identify an evasive malware sample?
- What percentage of the malware samples that you analyze are evasive?
- Is there anything that you would like to change in the process of identifying an evasive malware sample?
- In your questionnaire you mentioned that you use both static and dynamic analysis. Can you explain why you use both and what causes you to switch from one to the other?
- Is your workflow documented or standardized?
- How did you come up with your current workflow?
- Do you hold a college degree? If so, what was your major, and did it help with malware analysis?
- Do you know if your workflow is similar to your coworker's workflow in your group?
- Can you walk me through your process of analyzing an already identified evasive malware sample?
- What are you trying to accomplish when analyzing an evasive sample? What information do you want to extract?
- Is understanding evasion tactics helpful or do you just want to bypass them?
- What are the challenges that you encounter in your workflow and what tools would help you overcome them?

Techniques for Handling Evasive Malware.

- How do you handle malware that uses code obfuscation?
- Do you consider fingerprinting a significant evasive tactic? If so, what steps do you take to mitigate it?
- How do you handle malware that employs timing-based evasion techniques?
- How do you handle malware that checks for user interaction? If you don't, why not?
- When doing dynamic analysis, are you concerned with the malware detecting that it's being executed in an analysis environment? If so, what steps do you take to mitigate this?
- Which evasive malware techniques do you consider to be the most challenging to analyze?

- Which evasive malware techniques do you consider to be the most time-consuming to analyze?
- Why do you think evasive malware remains challenging to analyze?

Use of Existing Tools for Malware Analysts.

- What tools do you use when analyzing an evasive malware sample?
- Which tool would you consider to be the most helpful in your analysis workflow and why?
- Which tool do you use but would you like to be improved?
- When was the last time you implemented a new tool into your workflow?
- Where do you find new tools?
- What qualities do you consider when selecting a new tool?
- Is there any limitation or challenge that you would like a tool to automate?

C Codebook

Below is our complete codebook in their corresponding categories. We provide a brief description of the codes that may need further explanation.

Participant Role.

- Job Description: current job title and job
- Experience: years of experience in malware analysis and previous jobs.
- Education

Organization.

- Escalation: When malware samples get sent between teams.
- Mentoring: If malware analysts mention having a mentor or mentoring another individual regarding malware analysis.
- Operational Analyst: Participants that mention they work at AV companies, as malware analysts, or as incident responders.
- Research Focus: When the analysts' objective reaches beyond detection and requires a more in-depth understanding of the malware's origin and purpose.
- Restrictions: Factors that restrict the analysis process.
- Resources Influence Workflow: When malware analysts' workflow is affected or benefited by specific resources within their organization.

Malware.

- Acquisition of Malware: the process of acquiring a potentially malicious program.
- Malware Context: information the analysts receive about the malware sample.
- Commodity Malware: any information mentioned about commonly viewed malware.
- Sophisticated Malware: any information mentioned about sophisticated malware.
- Example of Malware: analyst mentions specific examples of malware samples, including the name of the family.
- Targeted Malware: when the analyst mentions a malware sample that avoids detection unless it is executed in its desired environment.
- Multiple Stages: analysts describes multi-stage malware
- Type of Malware Informs Analysis Process: a case when a participant's process changes based on the malware sample.
- Programming Language

Analysis Workflow.

- High-Level Workflow: description of the analysis workflow used by analysts to analyze any malware sample.
- Switch Trigger Between Static and Dynamic Analysis: what causes analysts to go back and forth between static and dynamic analysis.
- Need for Automation: a process that analysts mention would benefit from automation.
- Standardized Workflow: if the analyst mentions they have a standardized analysis workflow.
- Non-Standardized workflow: if the analyst mentions they do not have a standardized analysis workflow.
- Suspicious Activity: signs that may provide analysts hints about the program's malicious behavior.
- Process of Generating Signatures: how malware analysts create signatures.
- Objectives: the objectives analysts have when analyzing a malware sample.
- Automated Triage: a malware analysis pipeline through which each sample undergoes automated processing.
- Useful Information from One Type of Analysis to the Other: specific information that analysts take from static analysis to use during dynamic analysis or vice versa.

Static Analysis.

- Static Analysis Preference
- Basic Static Analysis: static analysis process such as checking for strings.
- Advanced Static Analysis: how analysts analyze the malware binary in a disassembler.
- Benefits of Static Analysis
- Limitations of Static Analysis
- Locating Suspicious Activity: how malware analysts locate suspicious activity in a disassembler.
- Where They Begin: how do analysts begin analyzing a sample in a disassembler.

Dynamic Analysis.

- Dynamic Analysis Preference
- Basic Dynamic Analysis: sandbox execution
- Advanced Dynamic Analysis: debugging process
- Sandbox configuration
- Sandbox Limitations
- Benefits of Sandbox
- Multiple Sandbox Execution: if they execute the sample multiple times and why
- Symbolic Execution: the use of symbolic execution
- Contributing to Sandbox: when analysts use new information from their analysis to improve sandboxes.
- Unpack Sample: process of unpacking a malware sample
- Use of Sandbox Report
- Bare metal: execute malware in bare metal systems.
- Emulation: analysts use emulation techniques.

Malware Evasion.

- Definition of Evasion
- Detecting Evasive Malware
- Frequency of Evasive Malware
- Most Challenging Evasive Techniques
- Most Time-Consuming Evasive Techniques
- Most Common Evasive Techniques
- Why Analyzing Evasive Malware Remains Challenging
- Correlation Between Malware Family & Evasive Technique
- Purpose of evasion technique
- Layers of Evasion
- Evasion-Based Signatures

Static Analysis Evasion.

- Static Analysis Evasion Techniques
- Frequency of static analysis evasion
- Bypassing Static Analysis Evasion: how malware analysts overcome static analysis evasion techniques.

Dynamic Analysis Evasion.

- Dynamic Analysis Evasion Techniques
- Frequency of Dynamic Analysis Evasion
- Bypassing dynamic analysis evasion: how malware analysts overcome dynamic analysis evasion techniques.

Implementation.

- Discovering New Tools: where analysts find new tools.
- Does Not Use New Tools Often: when analysts mention that they do not use new tools often.
- Willingness to Implement New Tools
- Qualities: when analysts mention qualities that they consider when deciding to use a new tool.

Tools.

- Malware Analysis Tools
- Most Helpful Tool
- Improvements for Tools: when analysts mention how existing tools can be improved.
- New Tool Idea: when analysts mention ideas for new tools.
- Internal Tools: when analysts mention tools made exclusively for their organization.
- Custom Scripts: when malware analysts describe how they create custom scripts.
- Open Source Vs. Custom Tools: when analysts compare open source with tools internal to the organization.
- Hard to Apply in Practice: when analysts mention that certain approaches are difficult to apply in practice.

D Participants

ID	Education	Yrs.	Analysis Preference
P1	N/A	10	Static Analysis
P2	IT	7	Static Analysis
P3	N/A	8	Static Analysis
P4	Math	10	Static Analysis
P5	CS	15	NA
P6	CE	11	Static Analysis
P7	IT	15	NA
P8	N/A	3	Static Analysis
P9	Computer Networks	3	NA
P10	Information Assurance	11	Dynamic Analysis
P11	CS & Math	8	Static Analysis
P12	CS	9	Static Analysis
P13	CS	6	Static Analysis
P14	CS	7	NA
P15	EE & Math	9	Static Analysis
P16	CS	27	NA
P17	CS	9	Static Analysis
P18	IT	5	NA s
P19	CS	12	Static Analysis
P20	CS	12	NA
P21	Digital Forensics	8	Static Analysis
P22	EE & Math	15	Dynamic Analysis
P23	CE	12	Static Analysis
P24	N/A	15	Dynamic Analysis

Table 3: Participants' Education (IT: Information Technology, CS: Computer Science, EE: Electrical Engineering, CE: Computer Engineering), Years of Experience, Analysis Preference, and Tiers [88] (Behavior: Goal of identifying potentially malicious activity, TTPs: Goal of understanding tactics, techniques, and procedures).