S.C.A.R.L.E.T: Stealth and Counter-measure metric for Anti-Reverse Engineering Technique Landscape

Pavan R Kashyap
PES University,
pavanrkashyap@gmail.com

Phaneesh R Katti
PES University,
phaneeshkatti@gmail.com

Hrishikesh Bhat P
PES University,
hriship3402@gmail.com

Pranav K Hedge
PES University,
pranavk1806@gmail.com

Prasad B Honnavalli
PES University,
prasadhb@pes.edu

Ethadi Sushma
PES University,
sushmae@pes.edu

Abstract

In the ever-evolving landscape of cybersecurity, reverse engineering, the process of dissecting software to understand its components and functionality, is fundamental for comprehending malware behavior and devising effective countermeasures. However, contemporary malware strains are employing increasingly sophisticated anti-reverse engineering techniques, aiming to obstruct the analysis process and impede a thorough understanding of their functionality. This escalation in sophistication forms a pressing challenge, compelling the need for a structured assessment and robust countermeasures. This paper proposes the Stealth and Counter-measure metric for Anti-reverse Engineering Techniques and Landscape (SCARLET) to effectively address this growing concern.

The SCARLET metric offers Stealth sCore for Anti-Reverse Engineering Landscape (SCARL) scores, designed to quantify the stealth level of evasion techniques within Windows malware samples. Leveraging real-world analysis, SCARL scores assist defenders in identifying suitable solution spaces for effective mitigation. The methodology involves a rigorous evaluation of evasion strategies, offering a systematic approach to understanding and countering these increasingly sophisticated anti-reverse engineering mechanisms. Empirical demonstrations of SCARLET's application to actual Windows malware instances underscore its effectiveness and enhance the credibility of this approach. Through the SCARL score, this research contributes to enhancing standardization of cybersecurity measures by providing a uniform and quantifiable assessment framework.

1 INTRODUCTION

This paper introduces the Stealth and Counter-measure metric for Anti-Reverse Engineering Technique Landscape (SCARLET), a comprehensive tool addressing challenges in reverse engineering. SCARLET classifies anti-reverse engineering techniques by assigning SCARL Scores (Stealth scores), aiding defenders in discerning their level of stealth and identification ease. The metric offers an exhaustive compendium of tailored solution spaces based on technique evasiveness, expediting reverse engineering and enhancing payload execution understanding. It's a valuable asset for threat intelligence and Incident Response (IR) teams, establishing a standardized framework for effective adaptation to emerging anti-reverse engineering techniques. Whilst there are multiple sub-categories of anti-reverse engineering techniques, this paper focuses mainly on:
A) **Anti-debugging techniques**
B) **Anti-sandbox techniques**
C) **Anti-VM techniques**

Focusing on these three key techniques, this paper presents a taxonomy of common attack vectors instrumental in crafting these methods for Windows machines. The metric is thoughtfully applied to these attack vectors, substantiated by weighted scores, and seamlessly integrated into existing solution spaces. Its adaptability ensures relevance over time, accommodating novel evasion techniques and corresponding solution spaces. This standardized approach to cybersecurity resilience helps institutions optimize resources, facilitate compliance, promote business continuity and reputation, while adapting proactive defensive measures.

2 CURRENT CHALLENGES

A) **Challenges in the Contemporary Landscape**

In the current landscape of malware threats [1], [6], [10], [12], anti-reverse engineering techniques have become increasingly evasive [8], [9]. Malware authors continually develop new strategies to hinder the monitoring and controlled execution of their malicious payloads. While these techniques serve as obstacles to understanding the core functionality of malware, they are not the primary problem themselves. This situation necessitates the urgent need for efficient solutions that do not overly deplete resources.

Defenders are faced with a challenge when selecting suitable response strategies for specific evasion techniques. The process often relies on intuition and lacks standardization, leading to inefficiencies and non-replicable outcomes.

B) **Diverse Evasion Techniques and Incomplete Solutions**

Current research trends [4] [15] focuses on elucidating solutions for specific evasion techniques, often overlooking the extensive array of evasive methods that can be employed in conjunction with or as alternatives to existing techniques. This diversity in evasion techniques remains largely unaccounted for in conventional research [17]. Moreover, the solutions proposed in such research tend to be either overly specific or overly general, lacking practical applicability in real-world scenarios.

C) **Resource Allocation Dilemma and Inflexible Metrics**

Companies and Incident Response(IR) Teams currently grapple with resource allocation challenges stemming from a lack of metrics to gauge the evasiveness of detected samples and their suitable response techniques.

Additionally, many existing metrics [14] [15], lack flexibility and adaptability in the face of the ever-evolving security landscape, rendering them less effective for integration into an organization's security posture.

3 PROBLEM SPACE

To comprehensively establish the metric, this paper initiated an exploration aimed at gaining an in-depth understanding of the challenges within the domain of anti-reverse engineering techniques. The initial focus was on identifying diverse techniques employed by malware to detect the presence of a debugger (by actively monitoring its own execution). Figure 01 presents a Control Flow Graph (CFG) diagram, illuminating some of the prominent methods used for the sole purpose of debugger presence detection. Each path in the CFG diagram preserves the primary objective of identifying the debugger's presence, yet they employ distinct stealth techniques to accomplish this task.

For instance, Path 01 employs the well-known DLL API call IsDebuggerPresent(), which straightforwardly informs defenders that a debugger presence check is underway. In contrast, Paths 05 and 06 involve more intricate maneuvers, such as inspecting specific directories, files, registry keys, and subkeys. These paths do not directly reveal the purpose of the technique, necessitating a deeper understanding of the malware's code segments and dynamic behaviors for defenders to discern the anti-reverse engineering technique's objectives.

This observation highlights that multiple methods exist for crafting a given anti-reverse engineering technique.Therefore, in order to assign stealth scores, it becomes imperative to comprehensively understand these various **attack vectors (AV)** currently in use.

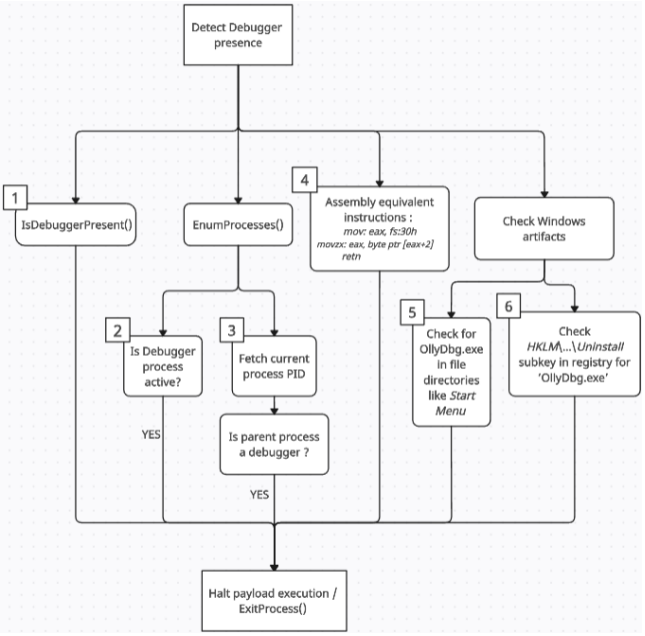


Figure 1: Control Flow Graph(CFG) highlighting evasive paths (1 to 6) employed to detect presence of debugger

3.1 Attack Vectors

Attack vectors are defined as the paths taken by the anti-reverse engineering technique [2], to evade detection. Some of the most prevalent attack vectors utilized by authors to construct their anti-reverse engineering techniques in contemporary cybersecurity landscapes are:

A) **Dedicated Dynamic Link Library Application Programming Interface (DLL API) Calls:** Certain DLL API calls are explicitly designed to ascertain the existence of specific tools or environments [11], [15], [16]. Malicious actors leverage these calls to identify the presence of debugging tools or virtualized environments. This attack vector is highlighted in Path 01 in Figure 01.

B) **DLL API Call Chains**: This attack vector involves the strategic chaining of multiple DLL API calls to gather information about the host system's processes and threads. Paths 02 and 03 in Figure 01 illustrate instances of such DLL API call chains.

C) **Assembly Code Equivalents**: In this technique, attackers embed assembly code equivalents of DLL API calls into malware samples to complicate the identification of them during reverse engineering. For instance, the assembly code equivalent for IsDebuggerPresent() is shown in Path 04 of Figure 01.

D) **Windows Management Instrumentation (WMI) Queries**: Malicious actors harness the capabilities of Windows Management Instrumentation (WMI) to craft malicious queries that extract information related to debugging and environmental conditions from the target system.

E) **Privilege Escalation and System-specific Contextual Data Acquisition**: This vector involves systematic querying and retrieval of system-specific parameters, often at escalated privileges such as CPU temperature and dynamic mouse movement, thus providing a crucial foundation for assessing the host's environment.

F) **Accessing File Artifacts**: Malware authors extend their reconnaissance by traversing various directories within the Windows system, gathering metadata and content data.As demonstrated in Path 05 of Figure 01, this meticulous search scans each directory for specific files, such as OllyDBG.exe [26], the executable associated with debugging tools.

G) **Accessing Registry Artifacts**: Malicious actors frequently probe the system's Registry, scrutinizing its keys and sub-keys to extract vital information. This includes identifying installed applications, programs configured to launch at startup etc. For example, in Path 06 of Figure 01, the inspection of the **HKLM/Software/Microsoft/CurrentVersion/Uninstall** key determines whether a debugger application is installed on the system or not.

Whilst these provide an exhaustive list of common attack vectors in use, there are several sub-attack vectors, that are usually employed in tandem with these major attack vectors to increase their evasiveness.

3.2 Sub-Attack Vectors

Sub-attack vectors (SAV) are those attack vectors that do not directly interact with artifacts or tools; they are strategically positioned in spatial proximity to the primary attack vector. Their primary objective is to impede or perplex defenders, diverting their attention and efforts away from the core attack. Some of the most prevalent sub-attack vectors employed are:

A) **Use of time-based delays**: Attackers use sleep() instructions to delay or stall the execution of the malware payload, thereby preventing sandboxes and debuggers from identifying the true intent of the malicious payload.

B) **Use of unconditional/conditional jump instructions (JMP)**: Attackers deliberately introduce several JMP instructions to different sections of the executable, under different labels, to confuse and mislead defenders.

C) **Introduction of rabbit hole operations**: Attackers deliberately drop rabbit hole code sections like spurious artifact checking etc. to mislead defenders into interpreting wrong and trivial details.

D) **Use of string matching** : Attackers use string matching to check if a regex pattern within their malware matches with artifacts associated with monitoring tools in the victim environment.

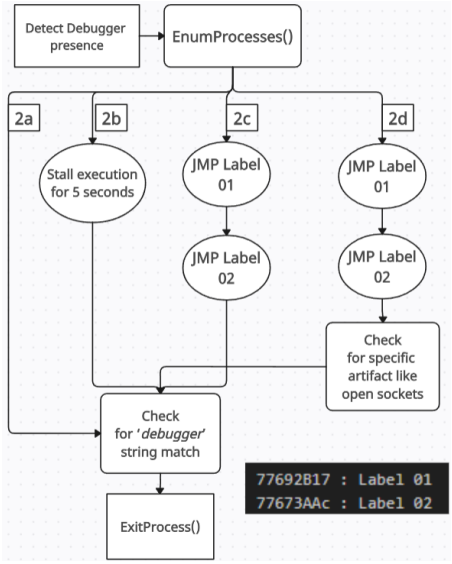


Figure 2: Control Flow Graph (CFG) highlighting how use of various SAVs makes the AV (DLL APIs) more evasive

E) **Propagating dead variables' results down the executable**: This redundancy is aimed at introducing bugs that will cause the program to crash if upper sections where these variables were first initialized get patched

F) **Other miscellaneous techniques**: This encompasses other techniques that are specific to certain anti-reverse engineering techniques like self-debugging etc.

In our exploration of sub-attack vectors and their impact on the evasiveness of primary attack vectors, we focus on Path 02 from our comprehensive diagram (Figure 01). This examination sheds light on how the incorporation of various sub-attack vectors can significantly enhance the evasive characteristics of an attack vector, specifically DLL API chains in this context.

Figure 02 serves as a visual representation of our findings. Path 02a, within this diagram, represents the conventional application of DLL API chains, devoid of any sub-attack vector elements.

Path 02b introduces a deliberate time delay of 5 seconds into the execution of the DLL API calls within the chain. This delay tactic serves to confuse dynamic analysis tools that monitor DLL API calls, by altering the expected timing patterns.

Path 02c employs an even more evasive technique—utilizing deliberate JMP (Jump) instructions to navigate between various blocks or labels within the executable. This constant jumping between different code sections keeps analysts and tools in a perpetual state of uncertainty regarding the DLL API chain's presence.

Path 02d combines the JMP instructions used in Path 02c with an artifact check. This artifact, unrelated to the core functionality of the attack vector, serves as a diversionary tactic. Such deceptive code fragments, known as "rabbit hole functions," are exceptionally evasive.

Our analysis demonstrates a clear progression in the evasiveness of the main attack vector, from the relatively straightforward Path 02a to the highly convoluted Path 02d, in comparison to Path 02.

It is important to note that while the attack and sub-attack vectors presented in this paper may not encompass every conceivable vector in existence, they effectively encapsulate the most prevalent templates employed by threat actors to construct their anti-reverse engineering techniques within the current cybersecurity landscape.

Understanding the problem space and the attack vectors paves way for a more inclusive SCARL score, that accounts for the attack vectors, rather than the attack techniques involved. Before, delving into the SCARLET metric, it is prudent to understand the solution space too, so solutions can be effectively mapped to different stealth score categories.

4 SOLUTION SPACE

In order to effectively combat and mitigate the identified attack vectors, we define the **solution space** as encompassing all comprehensive and viable defensive measures tailored to counter the attack vectors outlined earlier, ensuring a thorough and robust defense against potential threats.

To organize the multitude of defensive measures, we categorize similar solutions into distinct **solution-types**. These solution-types are designed to address overarching attack vectors, encapsulating a range of specific defensive strategies within each category. It is imperative to emphasize that our focus is on presenting solution-types aligned with the broader attack vectors, rather than delving into specific attack techniques at this juncture:

A) Execution Speed-up (used by sandboxes): Sandboxes monitor the execution speed and can accelerate the execution to counteract attempts by anti-reverse engineering techniques to slow down or stall the analysis. Sandboxes can manipulate the perception of time within the virtual environment, either by speeding up the clock or advancing timestamps, to counter any delays introduced by time-based anti-reverse engineering techniques.

B) Memory Manipulation: Dynamic data transformation involves altering critical data and instructions in the memory at run-time so that the original payload can continue its execution without any hindrance, when it is being monitored by a tool/environment. One such technique that falls in this solution-type involves manipulating the memory sections of the Windows machine on the debugger whilst debugging the execution of the malware sample. Introducing NOPs(No Operations) to effectively skip certain code sections or API calls is another technique that is commonly used by defenders to prevent anti-reverse engineering techniques from carrying out their intended checks.

C) Register Changes: Preventing malware from setting hardware breakpoints is crucial in hindering evasion attempts. Resetting the **CONTEXT_DEBUG_REGISTERS** flag and clearing debug registers (**DR0-DR7**) effectively obstructs malware from using hardware breakpoints.

D) CFG Diagrams for Control Flow Observations: CFG diagrams help in visualizing the program's control flow. Observing and analyzing the CFG aids in identifying what paths taken by the malware sample are legitimate and what paths are rabbit-holes, essentially helping defenders route around those rabbit-holes to focus on the main attack vector.

E) Data Spoofing: Data spoofing involves modifying or fabricating data to mislead anti-reverse engineering techniques. Common strategies include checksum alterations and tampering with data integrity. Anti-reverse engineering techniques are deceived into believing that they are running on host machines, thereby revealing the true payload in the controlled environment.

F) Environment Virtualization: Creating virtualized environments allows defenders to study anti-reverse engineering technique's behavior in a controlled and isolated setting. By virtualizing the environment, security professionals can observe and analyze malware actions carried out without risking the host system's security.

G) Resource Hiding and Spoofing: Anti-reverse engineering techniques often rely on specific system artifacts for their activities. Hiding or spoofing these artifacts, such as registry keys, files, or network-related identities, can confuse the anti-reverse engineering technique, and allow the execution of the malware, thereby helping defenders understand the dynamic behaviors of the sample.

H) API Hooking: Proactive defense through API hooking involves intercepting API calls and manipulating the responses they generate, to mislead anti-reverse engineering techniques into believing that there are no tools/environments that are monitoring them.

I) Other General Defensive Techniques: Other general solutions to counter anti-reverse engineering techniques include kernel patching, analysing assembly code, strict User Access Controls (UAC), leveraging Least Privilege Principle, Data Execution Prevention (DEP) in Memory, etc.

5 SCARLET WORKING

The crux of the SCARLET metric lies in its categorization of these anti-reverse engineering techniques into three principal brackets, each distinguished by unique attributes.

A) **Mildly Evasive Bracket**: This category houses anti-reverse engineering techniques employing attack vectors characterized by their relative ease of comprehension and detection.

B) **Moderately Evasive Bracket**: In contrast, the moderately evasive bracket is designated for techniques that leverage attack vectors posing a greater challenge to decipher and understand.

C) **Highly Evasive Bracket**: The third bracket is reserved for techniques employing attack vectors that delve deep into a system's low-level architecture, initiating attacks that are particularly intricate to uncover.

5.1 Key Attributes of Brackets

Each of these brackets boasts three key attributes:

A) **SCARL Score Range**: A defined range of SCARL scores, serves as the benchmark for categorizing anti-reverse engineering techniques into the respective brackets. SCARL scores are assigned to these techniques based on their evasiveness levels and then mapped into one of the three brackets.

B) **Forward-Thinking Buffer Space**: Recognizing the evolving nature of cyber threats, buffer space is thoughtfully allocated at the upper limits of each bracket. This strategic provision accommodates the integration of new attack vectors with heightened stealth attributes in the future.

C) **Diverse Solution-Types**: Within each bracket, various solution-types exist to address specific challenges. These solution-types span various SCARL score levels, sometimes overlapping to cater to different nuances within the bracket. Furthermore, each solution-type comprises multiple solution-bands, offering tailored solutions at distinct SCARL score levels.

By delineating these brackets and their associated attributes, the SCARLET metric seeks to bridge the gap between qualitative and quantitative threat models. This approach facilitates seamless integration into existing organizational threat models by introducing SCARL scores and solution-types, thereby enhancing the precision and adaptability of cybersecurity practices.

5.2 Quantifying SCARL score ranges

The primary task when establishing the SCARLET metric was to assign SCARL score ranges to each bracket and subsequently extend these scores to encompass attack and sub-attack vectors . The initial consideration revolved around allocating a SCARL score range of 10 for each bracket. This notion was rooted in the assumption that each bracket would house an equal number of attack vectors. However, upon closer examination, as detailed in prior research [15], an inherent disproportion among the brackets was discerned. Specifically, the bracket accommodating mildly evasive techniques exhibited a substantial abundance, while the moderately evasive bracket contained fewer instances, reflecting the intricacies involved in crafting and deploying such techniques. In stark contrast, highly evasive techniques remained scarce and sporadic, primarily due to the formidable challenges of innovating low-level architectural methods.

Recognizing the imbalance, the SCARL score ranges were adjusted accordingly, keeping the mildly evasive bracket's range as 10 and allocating a range of 8 to moderately evasive (a size smaller than mildly evasive) and 6 (smaller still) to the highly evasive bracket. While a decreasing trend might have prompted considerations of reducing the highly evasive bracket to a score of 4, we opted to retain it at 6. This decision was driven by our anticipation of potential future techniques, such as those involving quantum encryption, which could pose formidable obstacles in the domain of reverse engineering.

This dynamic yielded the following SCARLET range ratio for each bracket:

Mildly Evasive : Moderately Evasive : Highly Evasive

10 : 8 : 6

This ratio served as the foundational framework for assigning scores to each bracket and subsequently delineating appropriate SCARL scores for individual attack and sub-attack vectors.

For the sake of comprehensibility, a **multiplicative factor** of **1** was chosen to augment this ratio, resulting in SCARL scores within a consolidated range of **24** (10 + 8 + 6) for the entire metric. In adhering to traditional conventions of numbering, the SCARL scoring system was initiated with a **base score** of **0**. Consequently, each bracket was allocated the following SCARL score ranges: **Mildly Evasive** (0 to 10), **Moderately Evasive** (10 to 18), and **Highly Evasive** (18 to 24).

5.3 Generating Attack vectors' SCARL scores

One fundamental premise of our work is that scoring individual attack vectors is more informative and practical than evaluating the techniques themselves. Attack vectors serve as foundational building blocks for a wide range of attack techniques. While the number of attack techniques can grow exponentially, the core attack vectors remain relatively stable. By focusing on these vectors, we enable defenders to categorize and prioritize their responses effectively.

The SCARL score for every attack vector is derived from two key factors:

A) **Ease of Understanding**: This factor assesses how readily a defender can discern the specific tools or environments targeted by an attack vector (attack vectors form the template for anti-reverse engineering techniques). Some attack vectors may explicitly reveal this information, while others require deeper analysis.

B) **Solution Mapping**: This aspect considers the types of mitigation strategies or solution-types that can be associated

Attack Vectors	SCARL Score
Dedicated DLL API Calls	01.00
DLL API Call Chains	
Simple	02.50
Complex	02.90
WMI (Windows Management Instrumentation) Queries	04.75
Checking Registry Artifacts	10.50
Checking User files	11.20
Checking Windows System Artifacts	11.40
Software Breakpoints Usage	18.00
Hardware Breakpoints Usage	18.50
Assembly Equivalents	19.50
Structured Exception Handling (SEH)	21.00

Figure 3: Attack Vector SCARL Score Table

with a given attack vector. It accounts for the nature of these solution-types and their potential overlaps with other attack vectors.

The attack vectors listed in the problem space are first mapped into one of the three brackets based on the ease of understanding. Subsequently once mapped into the bracket, they are **relatively graded** within the bracket based on the solution-types available and their relative ease of understanding the tools and environments they touch, within a given bracket. It's important to note that the process of relative grading within brackets has been a subjective decision of the authors.

A SCARL score of **0 indicates that no evasion techniques** are present. Conversely, a score of **24 represents extreme evasion**, rendering it exceptionally challenging for a defender to decipher the targeted tools or environments.

A comprehensive list of SCARL scores assigned to all attack vectors is indicated in Figure 03, presented with precision to two decimal places (d.p).

DLL APIs are the most comprehensible among the set of attack vectors (they are mapped to the mildly evasive bracket). These **dedicated DLL APIs** are supported by a wealth of available solution-types, rendering them highly manageable and earning a SCARL score of **01.00**.

DLL API chains, defined by the chaining of several DLL APIs are categorized into the mildly evasive bracket. Simple DLL API chains, defined by their reliance on documented Windows API calls, possess an inherent transparency due to readily accessible information. In contrast, complex DLL API chains, marked by the inclusion of custom API calls, require defenders to delve into the specifics of these custom DLL APIs, adding a layer of complexity to their mitigation. To reflect this, we assign SCARL scores of **02.50 and 02.90 to simple and complex DLL API chains**, respectively. It's worth noting that both categories share overlaps with the solution-types of memory manipulation, which curtails the disparity in their scores.

Moving to more intricate attack vectors, we find that techniques involving the manipulation of registry artifacts, user files, and Windows system artifacts obscure the precise intent behind their actions making them constituents of the moderately evasive bracket. Consequently, we attribute SCARL scores of **10.50 for Registry artifacts**, **11.20 for User files**, and **11.40 for other Windows system artifacts**. These scores are closely aligned, reflecting the similarities in their solution-types.

The level of evasion escalates considerably when dealing with attack vectors such as Assembly equivalents and Breakpoints (software and hardware), which is why they are mapped to the highly evasive bracket. These techniques introduce formidable complexities, as defenders grapple with deciphering assembly code or understanding the intricate deployment of breakpoints to obstruct debugging and monitoring processes. Their SCARL scores reach as high as **19.50 for Assembly** and **18.00-18.50 for Breakpoints**.

Path	Primary Attack Vector	SCARL Score
1	Dedicated API Calls	01.00
2	Simple DLL API Call Chains	02.50
3	Simple DLL API Call Chains	02.50
4	Assembly Equivalents	19.50
5	Checking User Files	11.20
6	Checking Registry Artifacts	10.50

Figure 4: Attack Vector SCARL Scores for all Paths Displayed in Figure 01

This scoring approach recognizes the nonlinear nature of the challenge; an Assembly equivalent with a score of 19.50 signifies a vastly distinct solution landscape compared to its predecessors.

Structured exceptions, earning the highest SCARL score of **21.00**, exhibit a unique modus operandi. These techniques capitalize on generating interrupts to execute malicious activities, rendering them exceptionally arduous to detect and mitigate within the current threat intelligence and solution framework.

Figure 04 provides a tabular representation of the employed attack vectors and their corresponding SCARL scores for evading debugger detection, as depicted in Figure 01.

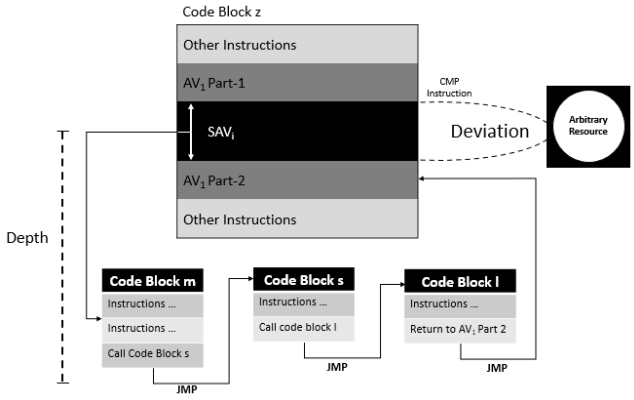


Figure 5: Diagram highlighting how Depth and Deviation Influence the Evasiveness of SAVs

5.4 Generating Sub-Attack Vectors' SCARL scores

Sub-attack vectors are strategically positioned within two portions of an attack vector to enhance the evasiveness of anti-reverse engineering techniques as seen in Figure 05. By introducing junk operations within the block where the two attack vector portions are, or by leading the main code block to jump to other junk code blocks, sub-attack vectors aim to misconstrue the understanding of the attack vectors. Similarly, by touching resources or artifacts that are not remotely associated with it, sub-attack vectors aim to stay stealthy and unrecognised by non-vigilant defenders. To effectively quantify the impact of sub-attack vectors and to provide a more comprehensive assessment, the inclusion of two additional parameters, depth, and deviation, alongside the existing parameters for scoring sub-attack vectors is proposed.

The four parameters governing our sub-attack vector scoring system are as follows:

A) **Depth**: This parameter evaluates the extent to which a sub-attack vector can manipulate the control flow within a program.

B) **Deviation**: Deviation measures the degree to which a sub-attack vector can divert defenders' attention from the main attack vector by introducing complexity and noise.

C) **Individual Contextual Complexity**: This parameter assesses how the presence of a specific sub-attack vector complicates the understanding of the code block and the overall attack vector.

D) **Solution Mapping**: Solution mapping accounts for the availability of solutions that can be used in conjunction with attack vector solutions to address issues arising from sub-attack vectors.

Implicitly considering these parameters which are all mutually exclusive, we derive SCARL scores for all sub-attack vectors, quantified within the range of 0 to 1 (up to two d.p). A SCARL score of 0 indicates minimal influence on the overall stealth of the primary attack vector, while a score of 1 signifies a substantial perturbation, significantly complicating the defender's task in devising solutions for the primary attack vector.

To aggregate the influence of all identified sub-attack vectors within the confines of attack vectors, we linearly accumulate their scores to determine the overall sub-attack vector score. These sub-attack vector scores are subsequently normalized to ensure that their linear combination within code block(s) does not disproportionately impact the overall sub-attack vectors' SCARL score (does not exceed 1), thereby preserving the metric's accuracy and consistency.

Mathematically, this normalization process is expressed as:

$$\max\left(\sum_{i=1}^{n} k_i * S.A.V_i\right) = 1 \tag{1}$$

where $S.A.V_i$ represents the ith sub-attack vector in the list and k_i represents the number of ith S.A.V used

Sub-Attack Vector	Sub-Attack SCARL Score
Use of Time Delay(s)	0 - 0.10
Use of Conditional and/or Unconditional Jumps (JMP)	$\sum_{j=1}^{n} \frac{0.075}{2^{(j-1)}}$, where j = # Jumps **0.15** (Maximum SCARL score when n = ∞)
Use of Rabbit Holes	$\sum_{k=1}^{n} \frac{0.15}{2^{(k-1)}}$, where k = # Deviations **0.30** (Maximum SCARL score when n = ∞)
Use of String Matching	0.12
Dead Variable Propagation	0.08
Miscellaneous Techniques	0 - 0.25
Total	**1.00**

Figure 6: Sub-attack Vectors SCARL Score Table

Figure 06 presents a tabular representation of the normalized scores attributed to various sub-attack vectors.
The use of jump (JMP) statements and rabbit holes are assigned a maximum score of 0.15 and 0.30 respectively. Since the number of recurrences of such techniques cannot be predetermined, we use the formula below to assign a dynamic score to such techniques:

$$\sum_{i=1}^{n} \frac{p}{2^{i-1}} \tag{2}$$

where p is the base SCARL score for that SAV and n is the a) number of JMPs for (un)conditional JMP introductions OR b) number of deviations for rabbit hole functions

$$\lim_{n\to\infty} \sum_{i=1}^{n} \frac{p}{2^{i-1}} = \frac{p}{1-r} \quad (r = 0.5) = 2 * p \tag{3}$$

The impact of introducing several JMP statements or rabbit hole functions tapers out as one continues to introduce it. A thousand JMP instructions become just as impactful as a million JMP instructions. This highlights that whilst we must account for their growths, it must not be accounted for either exponentially or geometrically with a growth factor greater than 1. Therefore, Equation 2, a geometric progression with r=0.5 (for simplicity in calculation) is devised. Where 'r' is the common ratio between successive terms in the expression a.k.a. the growth factor.

Even in the case of infinite depth (infinite JMP functions or infinite deviations), the overall score will not cross 2*p, thereby aligning with our initial argument of the JMPs and rabbit holes losing their impact significance as they are introduced in abundance. Defenders need not calculate the progression's summation in great detail, an estimate of the JMPs observed or the deviations seen can be promptly used to generate an approximate working score for the corresponding SAVs.

Taking these sub-attack vector SCARL scores into account, a holistic SCARL score for a given anti-reverse engineering technique is derived from a composite of the attack vector score and the overall sub-attack vector score. It can be mathematically represented as follows:

$$SCARL(M_x) = A.V_i + \sum_{j=1}^{n} k_j * S.A.V_j \tag{4}$$

In this equation, M_x denotes the xth anti-reverse engineering technique employed by the malware sample 'M', $A.V_i$ represents the 'i'th attack vector employed , $S.A.V_j$ represents the 'j'th sub-attack vector utilized and k_j represent the number of 'j'th SAV utilized by the same anti-reverse engineering technique.

While sub-attack vectors indeed contribute to the overall stealth of the attack vector, they do not significantly alter the inherent stealth score of the attack vector itself, regardless of the number of sub-attack vectors in play. This limitation is attributed to the fact that the technique's definition is primarily determined by the attack vector rather than the sub-attack vectors. Therefore, sub-attack vectors are appropriately normalized and capped at 1, preventing an excessive escalation of the SCARL score and any potential misplacement within different solution-types on the SCARLET Metric.

Figure 07 provides a comprehensive overview of the SCARL scores associated with each path illustrated in Figure 2. It's imperative to emphasize that this relative grading serves as a flexible framework. Organizations are afforded the liberty to customize and fine-tune this process to suit their specific needs. They can employ proprietary algorithms and black-box functions (denoted as f() in this context) to derive SCARL scores for attack and sub-attack vectors based on data culled from their diverse threat intelligence sources. By utilizing state-of-the-art machine learning techniques and leveraging their own insights, organizations can craft SCARL scores tailored to their unique contextual nuances.

Path	Sub-attack Vector	Final SCARL Score
1	Only Primary Attack Vector (Simple DLL API chain) NO Sub-attack Vector	**2.50** (Ref. Fig 4)
2	Use of Time Delay(s) Simple (5 Seconds)	2.50 + 0.02 **=2.52**
3	2x Jump Statements (JMP)	2.50 + 0.075 + (0.075/2) **≈ 2.61**
4	2x Jump Statements (JMP) - (Path 3) + Rabbit Hole	2.61 + 0.15 = **2.76**

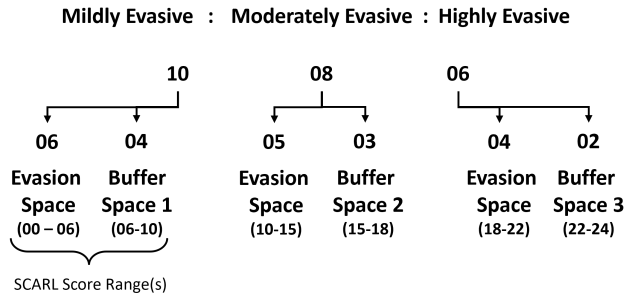
Figure 7: Final SCARL Score for all Paths discussed in Figure 02

5.5 High-level overview of SCARLET Metric

After successfully placing all the defined attack vectors into one of the three brackets, the best solution-types will now be placed into each bracket. Each solution-type, catering specifically to the attack vectors in that bracket will strategically be placed in the metric at different heights.

To accommodate for future attack vectors and novel solution techniques we decide to reserve space in each bracket as a buffer space.Each bracket is divided into two sections: the evasion space, that constitutes the lower end of the bracket and the buffer space, that constitutes the upper end of each bracket. No attack vectors or solution-types stated in the problem and solution-space are mapped to those buffer sections.

The three buffer spaces created inside each bracket are : **Buffer space 1 (6 to 10 SCARL scores), Buffer space 2 (15 to 18 SCARL score), and Buffer space 3 (22 to 24 SCARL score).**



Buffer spaces serve as transitional zones, accommodating attack vectors or solution-types that fall in between adjacent levels of evasiveness. This prevents abrupt categorization shifts and adds granularity to the metric. The inclusion of buffer spaces recognizes that not all elements within a bracket

are identical in terms of their evasiveness or complexity. This finer distinction enhances the metric's utility by aiding defenders in making more informed decisions when selecting mitigation strategies.

Figure 08 shows the high-level view of how the metric would appear once the SCARL scores, buffer spaces, attack vectors and solution-types are mapped onto the SCARLET metric.

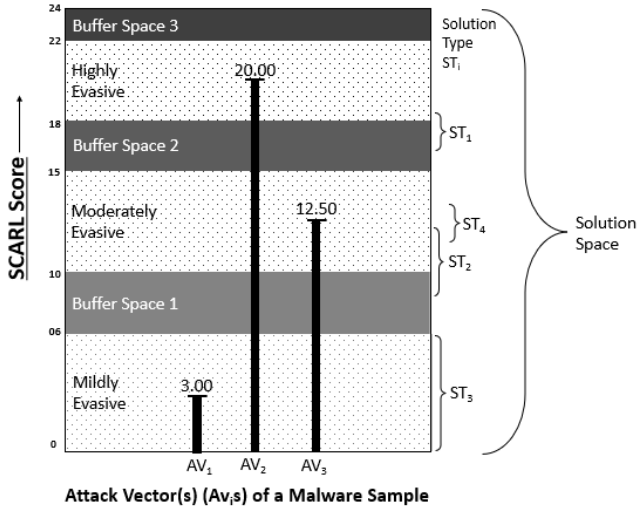


Figure 8: High-Level View of SCARLET Metric where S.T is Solution Type

Throughout the rest of the paper, whenever we refer to brackets, we will be considering only the evasion space and ignoring the buffer space accordingly.

5.6 Mapping solution-types to SCARLET

The solution-types mapped to each evasion bracket are listed below:

A) **Mildly Evasive Bracket (SCARL score varies from 0 to 6)**: Solution-types associated with this bracket comprise Memory Manipulation via Monitoring, encompassing techniques such as NOP introduction, register manipulation, and memory value modification. Most of these solutions are relatively straightforward to implement.

B) **Moderately Evasive Bracket (SCARL score varies from 10 to 15):** Solution-types linked to this bracket involve Resource Hiding, Data Spoofing, and Emulation of Virtual Environments. These solutions necessitate manipulations or alterations to system artifacts.

C) **Highly Evasive Bracket (SCARL score varies from 18 to 22):** Solution-types associated with this bracket encompass API hooking, kernel patching, and low-level manipulations.

Each bracket encompasses several solution-types, with multiple solution-types overlapping inside the bracket and between brackets.

The corresponding mapping of the attack vectors, their SCARL scores and the corresponding solution-types are highlighted in the Figure 09.

	Attack Vector	Base SCARL Score	Capped SCARL Score	Solution Type
Mildly Evasive	Dedicated DLL API Calls	01.00	02.00	Memory Manipulation
	DLL API Call Chains			
	Simple	02.50	03.50	
	Complex	02.90	03.90	
	WMI Queries	04.75	05.75	
Moderately Evasive	Checking Registry Artifacts	10.50	11.50	Data Spoofing Resource Hiding Resource Spoofing Environment Virtualization
	Checking User Files	11.20	12.20	
	Checking Windows System Artifacts	11.40	12.40	
Highly Evasive	Software Breakpoints	18.00	19.00	Memory Manipulation Data Spoofing
	Hardware Breakpoints	18.50	19.50	
	Assembly Equivalents	19.50	20.50	Low-Level Architecture Understanding
	Structured Exception Handling (SEH)	21.00	22.00	Code Signing API Hooking

Figure 9: Attack Vectors with SCARL Score Range and Corresponding Solution type

5.7 Need of Solution-bands

The effective SCARL score for an anti-reverse engineering technique results from the cumulative impact of all sub-attack vectors employed alongside the main attack vector. This implies that while we can employ a set of solutions, denoted as S, to address the main attack vector, we cannot use the same set of solutions directly to tackle the challenge posed by the main attack vector plus k sub-attack vectors. Consequently, it becomes imperative to select different solution sets that fall within the same solution-type for a particular attack vector.

As new sub-attack vectors are introduced(as the overall sub-attack SCARL score approaches 1)within two attack vector code portions, solutions aimed at solving the attack vector itself, become more and more limited and specific. In other words, the approach shifts towards addressing the impact of sub-attack vectors first before tackling the solution to the main attack vector itself. This evolution in specificity can be observed in Figure 10, which demonstrates how solutions become increasingly precise within a given solution-type as sub-attack vectors accumulate. It also highlights how solutions get more and more limited as the overall sub-attack SCARL score approaches 1.

This clearly highlights how there is an evident distinction of solutions within a given solution-type itself and therefore relying only on the solution-types cannot help defenders identify the best solutions to use to tackle the anti-reverse engineering techniques once they have mapped them onto the SCARLET metric.

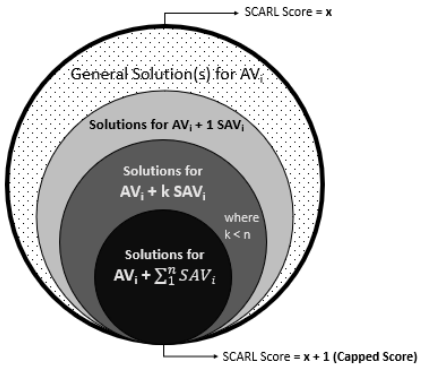


Figure 10: Solution type for 'i'th Attack Vector (AV_i)

5.8 Solution-bands

Relying solely on solution-types can hinder defenders in effectively countering Advanced Reverse Engineering Techniques (ARETs). To address this challenge, we propose the introduction of solution-bands within each solution-type, each associated with specific SCARL score ranges.

Solution-bands can be thought of as compact repositories that contain solutions tailored to specific SCARL score ranges. The solutions housed in a particular solution band can
A) Belong to two different solution-types.
B) Belong to a certain attack vector k (without any sub-attack vectors).
C) Belong to a certain combination of attack vector k and n sub-attack vectors.
D) Belong to another attack vector or attack-sub attack vector combination.

Creating effective solution-bands involves defining the SCARL score ranges associated with them. Defenders start by setting a certain width for these bands and filling them with solutions obtained from threat intelligence sources. During this process, defenders have the flexibility to merge closely-related solution-bands by expanding their width or, conversely, divide solution-bands containing dissimilar solutions into smaller, more focused segments.

The primary aim of ensuring that all scores are set upto 2 d.p is to ensure that the solution-bands being crafted here do not get too granular. The aim is to ensure that solution-bands are reasonably well sized, so that they can accommodate all the various solutions that cater to SCARL scores in that particular band or range.

While some critics may argue that introducing solution-bands housing solutions from different attack vectors makes retrieval less efficient, our approach harmonizes and streamlines an extensive solution space into well-defined solution-types and solution-bands. These categorizations represent efficient channels for deploying solutions. By including solutions that exhibit slight variations from a particular attack vector, we cover the full spectrum of solutions closely associated with that specific vector without making the bands overly specific. Furthermore, by excluding unnecessary solution categories, we ensure that overly broad solutions are not included in our metric.

This refined strategy ensures that defenders have a comprehensive yet manageable framework for selecting precise solutions to address the complex landscape of ARETs.

5.9 Mapping solution-bands to SCARLET

To understand the mapping of solution-bands to the SCARLET Metric, we turn our attention to Figure 11, which highlights how solution-bands would hypothetically map onto on the SCARLET metric.

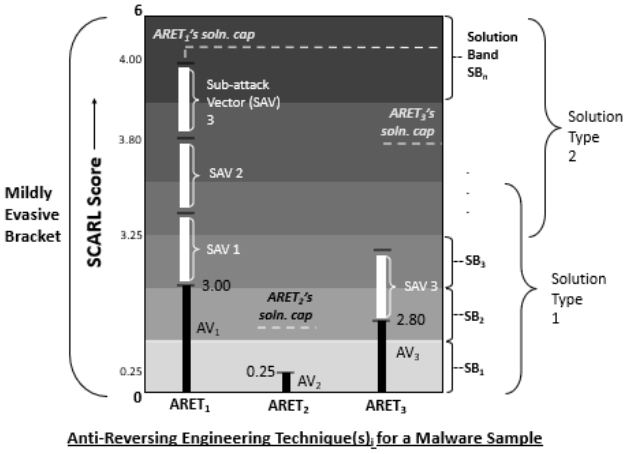


Figure 11: SCARLET Metric showing SCARL scores and solution-bands for different Anti-reverse engineering techniques

Solution Band 1 caters to solutions associated with Attack vector 2 (employed by ARET 2). Solutions associated with attack vector 2 and k sub-attack vectors that lie within this solution-band's SCARL score, are also mapped into this solution-band. Attack vector 1 (employed by ARET1) and Attack vector 3 (employed by ARET 3) have base scores that map beyond this solution-band. Therefore, no solutions pertaining to AV1, AV3 and their sub-attack vector linear combination lie in solution-band 1.

Solution band 3 holds solutions to Attack Vector 1, Attack vector 1 + SAV1 and Attack vector 2+ SAV3. This reinstates the characteristics of the solution-band described previously. An interesting point to note is that solutions pertaining to just Attack vector 2 or Attack vector 1 + SAV1 + SAV2 do not

pertain to solution band 3.

The concept of the maximum cap for each attack vector signifies the highest achievable SCARL score for that attack vector. In this context, it suggests that all solution bands with SCARL scores exceeding 4.00 do not contain any solutions applicable to Attack Vector 1 or any of its sub-attack vector combinations. This highlights that it is futile to look at solution bands that exceed SCARL scores of 4.00 for Attack vector 1 and its sub-attack vector combinations.

By establishing and populating the solution-bands in the SCARLET metric, defenders can very easily go about tackling new malware samples they encounter. This consolidated metric now effectively serves the intended purpose of not only quantifying attack vectors but also leading them towards the right solutions too.

5.10 Maintenance and Evolution of the SCARLET Metric

To adapt the SCARLET metric to evolving threats and solutions, several strategies are in place:

A) When a new attack vector is identified, it is integrated into the metric by assigning it a score within one of the existing brackets (evasion space or buffer space).

B) New solution-types, when discovered, are mapped to relevant solution-bands, expanding the diversity of available solutions.

C) Emerging solution-types for newly identified attack vectors are appropriately mapped to corresponding solution-bands, typically spanning from the base to maximum cap of attack vector scores.

D) Newly surfaced sub-attack vectors can be included in a miscellaneous category or have their scores normalized and integrated into the current framework.

E) If some attack vectors become obsolete, only solutions within the specific bands for those vectors can be pruned and archived, simplifying the elimination process while preserving solutions applicable to other vectors. Should those old solutions be needed in the future because of an attack vector resurgence, they can simply be added back to the metric's appropriate solution bands.

F) Changes to a solution-band do not affect adjacent bands, minimizing the ripple effect of modifications and facilitating the metric's adaptability to evolving threats and solutions. This approach maintains clarity and precision in scoring and classification, ensuring a robust framework for anti-reverse engineering techniques.

6 METRIC IN ACTION

The effectiveness of the SCARLET Metric in guiding defenders to optimal solutions hinges on its meticulous configuration with solution-bands prior to application. To comprehensively evaluate our metric, we embarked on a process of populating it with solutions across various SCARL scores and assessing its practicality.

In this endeavor, we gathered external intelligence on a range of anti-reverse engineering techniques, assigned SCARL scores to them, and then diligently allocated their corresponding mitigation solutions to the appropriate solution-bands. To maintain simplicity and consistency, our solution-bands were uniformly fixed at a width of 0.2, resulting in bands ranging from 0-0.2, 0.2-0.4, and so forth, up to 24.

The sources from which we curated this data, including references such as [1], [2], [7], [12], were carefully selected due to their invaluable collection of disassembled or debugged code related to real-world malware samples. Furthermore, these sources contained specific mitigation strategies aimed at countering these techniques. The availability of disassembled and decompiled code enabled us to compute SCARL scores, while the accompanying solutions facilitated the population of solution-bands. Remarkably, these solutions, although originating from diverse sources, remarkably aligned with the solution-types we delineated in Figure 09. This alignment reaffirmed the credibility of our chosen solution-types for tackling specific attack vectors. For

ARET	Attack Vector(AV)	SCARL Score	Specific Solution	Reference(s)
IsDebuggerPresent	Dedicated API	1.00	Use Scyllahyde or Memory manipulation	[1][2][3][22] [12][20]
BlockInput(), GetCurrentProcessId()	DLL API Chains	2.50	Fill with NOPs	[1][2] [12][20]
Thread hiding	DLL API Chains	2.90	Hook multiple APIs	[1][2][22] [12][20]
User and System Name Check	System Artifacts	11.20	Change the Default User and System name	[1][2][11]
No. of active CPUs	System Artifacts	11.40	Using Virtual Machines to Change the no. of Cores	[1][2][11][5]
rdtsc	System Artifacts	11.40	Using Virtualization (QEMU) to manipulate TSC Flag	[2][3][5][12]
CPUID check	System Artifacts	11.50	Change the ".vmx" File	[1][2][12]
INT3	Software Breakpoints	18.00	Fill with NOPs	[1][2][3] [12][22]
Hardware Breakpoints	Hardware Breakpoints	18.50	Hook and Change specific registers	[1][2][22]
Vectored Exception Filter	Exception Handler	21.00	Use Scyllahide or Patch opcodes that cause exception(s)	[1][2][22]

Figure 12: Consolidated corpus to populate the SCARLET metric

the population of the SCARLET Metric, we turned to diverse malware samples, including Raspberry Robin, Alkhaser, and Beep, each known for employing multiple evasion techniques. Raspberry Robin is a malware dropper that resembles a worm and is widely used in malware campaigns. Alkhaser is a public benign malware that is a bundle of various anti-reverse

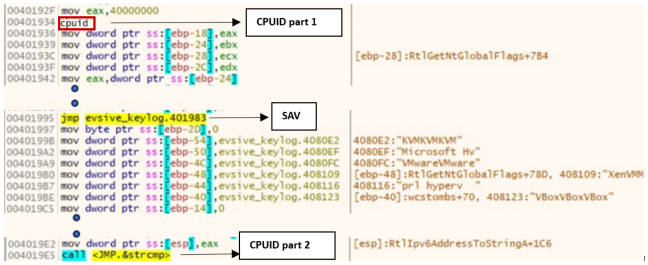


Figure 13: CPUID attack technique as seen in keylogger's disassembled code

engineering techniques. Beep is a malware that comprises of dropper, injector and a payload which employs anti VM and debug checks at each point of its execution. These samples served as ideal candidates for building the metric table. In Figure 12, we present a comprehensive overview of the attack vectors, their corresponding SCARL scores, the associated solutions, and the references that have been meticulously collated and mapped onto our SCARLET Metric.

It's important to note that, while our empirical research offers valuable insights, organizations undertaking a more extensive and thorough endeavor should ensure the comprehensive population of their SCARLET Metric. Nevertheless, our focused research serves as a robust starting point.

Having completed the population of the metric, the next step involved selecting a new malware sample from the wild. We assigned SCARL scores to this sample and evaluated the solutions mapped to those scores to assess the metric's efficacy. In this scenario, we chose a keylogger (SHA-256: *8287c4d4c710476fbdf97ed36a9a373a5b261a5b72bc2f00e2a890980189daf1*), identified as a spyware which logs the keyboard inputs, was selected from Virustotal [25] and the reports were generated from Hybrid Analysis [24]. Keyloggers in the wild frequently employ diverse evasion techniques to maintain stealth, making them ideal subjects for evaluating our metric.

A meticulous static analysis of the chosen sample revealed the presence of a CPUID check as the anti-VM technique, as highlighted in references [1], [2], [5], [11], and others. CPUID, an instruction common in X86 and X86-64 processors, serves to extract information about the processor's underlying ID, effectively examining system artifacts. Our metric, as depicted in Figure 09, classified the CPUID technique as moderately evasive, as it was touching system artifacts. This assessment is further illustrated in Figure 13, which showcases the specific code snippet employing this technique.

Attack Vector technique : Checking System Artifacts
Attack Vector score : 11.40

Notably, this attack vector comprises two distinct segments: one focused on executing CPUID instruction (Attack vector part 1) and another on comparing the return values stored in ECX and EDX registers with hard-coded hypervisor names (Attack vector part 2). If the comparison returns true, then the executable stops its execution, reasserting the point that it is an anti-VM technique.

We observe a JMP instruction introduced between these two attack vector parts in Figure 14, highlighting that there is an unconditional JMP in play (our sub-attack vector).Deeper examination uncovered that the presence of two JMP instructions enables the transfer of control to a distinct label before circling back to the same code block, indicating a depth parameter to be 02.

Sub-attack vector technique: Use of unconditional JMP instructions
Sub-attack vector score :

$$= 0.075 + \frac{0.075}{2} = 0.1125 \approx 0.11(2\ d.p.) \tag{5}$$

Final SCARL score for CPUID technique in Keylogger = Attack Vector Score + Sub-attack vector Score

$$= \mathbf{11.40 + 0.11 = 11.51} \tag{6}$$

This score was then mapped onto the SCARLET metric, placing the attack vector in the solution-band ranging from **11.40 to 11.60**. This solution-band already contained solutions tailored to CPUID-related attacks (based on population done in the previous stage). Given the proximity of the calculated SCARL score to the 11.50 solution, specifically concerning the manipulation of the .vmx file, we identified this as the optimal solution for the present scenario.
By slighty tweaking the populated solution [1], [11],we were able to quickly solve the given problem without having to worry about what solution-types to consider and eliminate to get to the best solution.

This expeditious and precise solution mapping allowed for the swift mitigation of the anti-reverse engineering technique, enabling us to focus on understanding and mitigating the primary payload of the Keylogger.

The metric was able to accurately suggest the solution to utilize, without heavily depending on a specific malware instance. Hence, this highlights that the proposed metric approach is consistent in addressing diverse evasion techniques present in diverse malware samples.

Our empirical assessment, using malware samples, demonstrated swift and precise identification of attack vectors and their optimal solutions. This efficient mapping not only substantiates metric's value as a versatile security metric, providing invaluable insights for defense strategies, but also validates the initial metric introduced in Figure 09.

7 ADVANTAGES

The SCARLET metric and its SCARL scores offer several key advantages-

A) **Precision Through Quantization**: SCARLET addresses a critical challenge in cybersecurity metrics by introducing quantization. While previous metrics [15] relied on qualitative assessments, SCARLET's quantization allows for precise differentiation between techniques within and across brackets. This objectivity is invaluable, enabling defenders to prioritize techniques with higher SCARL scores, thereby enhancing the efficiency of threat mitigation.

B) **Seamless Transition**: SCARLET adopts a three-bracket scheme and defines SCARL boundary scores for each bracket, facilitating a smooth transition from qualitative to quantitative metrics [27]. This approach ensures compatibility with existing threat assessment models [14], making it easier for organizations to embrace and implement the metric.

C) **Ease of Calculation**:Calculating the SCARL score while reverse engineering a malware sample simplifies the defender's task. Once computed, defenders can readily refer to the SCARLET metric for optimal solutions, saving valuable time and resources. The use of straightforward mathematical computations minimizes the need for excessive manpower in this activity.

D) **Customization for Specific Threat Intelligence**: SCARLET's flexibility shines through its ability to accommodate customization of SCARL scores to align with an organization's unique threat intelligence.This adaptability ensures that the metric remains in sync with evolving security requirements and internal threat assessments.

E) **Efficient Solution Management** : In the realm of SCARLET's capabilities, defenders are empowered to populate their solution-bands with a comprehensive array of defensive strategies that can originate from both internal development efforts and external sources such as websites and threat intelligence platforms. This approach eliminates the arduous task of extensively documenting and sifting through solutions when confronted with a threat. Instead, defenders can conveniently turn to the solution-band database corresponding to the relevant SCARL score. This streamlined process enhances defenders' agility and responsiveness, enabling them to swiftly identify and implement appropriate countermeasures with precision.

8 FUTURE WORK

In the realm of future research, several key areas demand attention to enhance the effectiveness and applicability of the SCARLET metric and SCARL scores for assessing anti-reverse engineering techniques.

A) **Community Collaboration**: Encouraging collaboration within the cybersecurity community to refine and expand the SCARLET metric collectively is pivotal. This collaborative effort can lead to a comprehensive and widely accepted framework for assessing anti-reverse engineering techniques.

B) **Integration with Security Tools**: Seamless integration of the SCARLET metric with existing security tools and platforms is essential, which can provide real-time threat assessments and automated mitigation recommendations.

C) **Advanced Scoring Techniques**: Exploring advanced scoring techniques that use machine learning and artificial intelligence to generate SCARL scores, can enhance the metric's precision in quantifying evasiveness. These methods can identify subtle evasion patterns and trends that might be challenging for manual analysts.

D) **Scalability**: Addressing the scalability of the SCARLET metric for large and complex malware samples is crucial. Research efforts should explore methods to efficiently apply the metric to extensive datasets.

These future research directions collectively aim to refine, expand, and further empower the SCARLET metric in its role of assessing and mitigating anti-reverse engineering techniques.

9 CONCLUSION

In the ever-evolving landscape of cybersecurity, the persistent challenge posed by anti-reverse engineering techniques demands innovative and quantifiable solutions. This paper introduced the Stealth and Counter-measure metric for Anti-reverse engineering techniques and Landscape (SCARLET) as a comprehensive and adaptable framework. SCARLET serves as a beacon of clarity, allowing defenders to navigate the intricate realm of evasion tactics with precision and confidence.

SCARLET metric focuses on quantifying evasiveness, providing a standardized and objective assessment that transcends the subjective nature of qualitative metrics. This quantization enables defenders to prioritize techniques effectively, enhancing the allocation of resources and the development of tailored countermeasures. Empirical validation of the metric demonstrates SCARLET's real-world effectiveness. Through the validation with diverse malware samples, SCARLET has proven its reliability and robustness in assessing the evasiveness of anti-reverse engineering techniques. This empirical evidence extends beyond SCARL score creation; it encompasses the practical implementation of mitigation strategies based on the metric's guidance.

While SCARLET represents a significant milestone in the assessment of anti-reverse engineering techniques, it also underscores the need for continued research and refinement. The dynamic nature of the field necessitates continuous innovation, ensuring that the defenders remain one step ahead of the adversaries. SCARLET is not merely a metric; it is a testament to our commitment to fortify the cyber defense ecosystem.

References

[1] Shavit Yosef, (2023, April 18), Accessed 2023, Oct 11, " RASPBERRY ROBIN: ANTI-EVASION HOW-TO and EXPLOIT ANALYSIS ", retrieved from *https://research.checkpoint.com/2023/raspberry-robin-anti-evasion-how-to-exploit-analysis/*

[2] LordNoteworthy. (Accessed 2023, Oct 11). Al-Khaser: An Advanced Anti-Analysis Toolkit [Software], GitHub: *https://github.com/LordNoteworthy/al-khaser*

[3] Zargarov, N. (2023, February 13). Beepin' Out of the Sandbox: Analyzing a New Extremely Evasive Malware. Minerva Labs.

[4] M. S. Islam, K. N. Khasawneh, N. Abu-Ghazaleh, D. Ponomarev and L. Yu, "Efficient Hardware Malware Detectors That are Resilient to Adversarial Evasion," in IEEE Transactions on Computers, vol. 71, no. 11, pp. 2872-2887, 1 Nov. 2022, doi: 10.1109/TC.2021.3068873.

[5] Pham Nep and Nguyen Cam, "A Research on Countering Virtual Machine Evasion Techniques of Malware in Dynamic Analysis," in *International Conference on Intelligent Computing and Optimization*, published in October 2022, pp. 585-596, ISBN: 978-3-031-19957-8, DOI: 10.1007/978-3-031-19958-5_55.

[6] M. Kim, H. Cho and J. H. Yi, "Large-Scale Analysis on Anti-Analysis Techniques in Real-World Malware," in IEEE Access, vol. 10, pp. 75802-75815, 2022, doi: 10.1109/ACCESS.2022.3190978.

[7] Lauren Podber, Stef Rand, Red Canary's Report on RaspberryRobin, "Raspberry gets the worm early", (2022, May 5), Accessed on 2023, Oct 11,https://redcanary.com/blog/raspberry-robin/

[8] De Gaspari, F., Hitaj, D., Pagnotta, G. et al. Evading behavioral classifiers: a comprehensive analysis on evading ransomware detection techniques. Neural Comput and Applic 34, 12077–12096 (2022),

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

[10] D. C. D'Elia, E. Coppa, F. Palmaro and L. Cavallaro, "On the Dissection of Evasive Malware," in IEEE Transactions on Information Forensics and Security, vol. 15, pp. 2750-2765, 2020, doi: 10.1109/TIFS.2020.2976559.

[11] Itamar Medyoni, (2020, Nov 16). Accessed 2023 Oct 11, "Malware Anti-VM techniques", Cynet, *https://www.cynet.com/attack-techniques-hands-on/malware-anti-vm-techniques/*

[12] Kim, Jong-Wouk Bang, Jiwon Choi, Mi-Jung. (2020). "Defeating Anti-Debugging Techniques for Malware Analysis Using a Debugger". Advances in Science, Technology and Engineering Systems Journal. 5. 1178-1189. 10.25046/aj0506142.

[13] H. Priya, K. Bhagavan, "Anti - Reverse Engineering Techniques Employed by Malware R," in Computer Science, published by Blue Eyes Intelligence Engineering & Sciences, 2019.

[14] A. Walker, M. F. Amjad and S. Sengupta, "Cuckoo's Malware Threat Scoring and Classification: Friend or Foe?," 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 2019, pp. 0678-0684, doi: 10.1109/CCWC.2019.8666454.

[15] Amir Afianian, Salman Niksefat, Babak Sadeghiyan, and David Baptiste, "Malware Dynamic Analysis Evasion Techniques: A Survey," arXiv preprint, arXiv:1811.01190, 2018, cs.CV.

[16] Chandra Sekar Veerappan, Peter Loh Kok Keong, Zhaohui Tang, and Forest Tan, "Taxonomy on malware evasion countermeasures techniques," *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, 2018, pp. 558-563, DOI: 10.1109/WF-IoT.2018.8355202

[17] 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*, published by the Association for Computing Machinery, 2017, pp. 2,

[18] J. Miljak, "An experimental study on which anti-reverse engineering technique are the most effective to protect your software from reversers' ', Dissertation, 2017.

[19] Jonathan A.P. 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)*, 2012, pp. 744-749.

[20] Peter Ferrie, "ANTI-UNPACKER TRICKS", in CARO Workshop, Amsterdam, 2008, *http://pferrie.epizy.com/papers/unpackers.pdf*

[21] Lorenzo Cavallaro, P. Saxena, and R. C. Sekar, "Anti-Taint-Analysis: Practical Evasion Techniques Against Information Flow Based Malware Defense,", published in 2007,

[22] Checkpoint Research, "Anti-debug Tricks", (Accessed 2023 Oct 11) *https://anti-debug.checkpoint.com/*

[23] Malware Bazaar, [online], (Accessed on 2023 Oct 12), Available: *https://bazaar.abuse.ch/*

[24] Hybrid Analysis, from CloudStrike Falcon, [online] (Accessed on 2023, Oct 11), Available: *https://www.hybrid-analysis.com/sample/8287c4d4c710476fbdf97ed36a9a373a5b261a5b72bc2f00e2a890980189daf1/652766a27f74aef78f0b0653*

[25] Virus Total, [online] (Accessed on 2023, Oct 12), Available: *https://www.virustotal.com/gui/file/8287c4d4c710476fbdf97ed36a9a373a5b261a5b72bc2f00e2a890980189daf1/detection*

[26] Ollydbg Documentation, [online] (Accessed on 2023, Oct 12), Available: *https://www.ollydbg.de/version2.html*

[27] Fawaz Rasheed, (2023, Sept 28), "Moving From Qualitative to Quantitative Cyber Risk Modeling", (Accessed on 2023, Oct 12), *https://www.securityweek.com/moving-from-qualitative-to-quantitative-cyber-risk-modeling/*