SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

April 2023, Vol. 66 142104:1–142104:19
https://doi.org/10.1007/s11432-021-3567-y

Framework for understanding intention-unbreakable malware

Tiantian JI[1], Binxing FANG[1], Xiang CUI[2*], Zhongru WANG[3], Peng LIAO[1] & Shouyou SONG[1,4]

[1]*Key Laboratory of Trustworthy Distributed Computing and Service (BUPT), Ministry of Education, Beijing University of Posts and Telecommunications, Beijing* 100876*, China;*
[2]*Cyberspace Institute of Advanced Technology, Guangzhou University, Guangzhou* 510006*, China;*
[3]*Chinese Academy of Cyberspace Studies, Beijing* 100010*, China;*
[4]*Beijing DigApis Technology Co., Ltd., Beijing* 100081*, China*

Received 21 December 2021/Revised 2 May 2022/Accepted 11 August 2022/Published online 27 March 2023

Abstract The anti-analysis technology of malware has always been the focus in the cyberspace security field. As malware analysis techniques evolve, malware writers continually employ sophisticated anti-reverse engineering techniques to defeat and evade state-of-the-art analyzers. Therefore, to prepare for unknown attacks, studying malware analysis techniques is insufficient. More importantly, we should study new anti-analysis techniques for malware. This paper expands the concept of anti-analysis malware and defines a type of intention-unbreakable malware (IUM). To help defenders fully understand and establish a defense system against the new security threats, this paper systematically discusses IUM from the perspectives of threat discovery, threat modeling, attribute analysis, and threat assessment; in addition, this study also provides the PoC implementations of IUM and demonstrates how attackers can use this type of malware to evade advanced security analysis, that is, accurate identification of target (class) victims and intention concealment in reaching non-target (class) victims. Finally, this paper provides several mitigation directions for combating IUM.

Keywords malware, unbreakable property, modeling mechanism, accurate identification, intention concealment

Citation Ji T T, Fang B X, Cui X, et al. Framework for understanding intention-unbreakable malware. Sci China Inf Sci, 2023, 66(4): 142104, https://doi.org/10.1007/s11432-021-3567-y

1 Introduction

Recently, fierce offensive and defensive arms races have led to the rapid development of various types of malware programs. To ensure the survivability and confrontation of malware, anti-analysis malware [1] and stealthy malware [2] have been developed and have become the preferred choices for attackers in actual offensive exercises.

Table 1 lists the typical anti-analysis methods and stealth-keeping methods commonly adopted by attackers. For example, encryption, obfuscation, and polymorphism [3, 4] are used to counter static analysis [5, 6]; and memory injection [7], macros, and PowerShell [8] are leveraged to achieve fileless attacks [9, 10]. Nevertheless, anti-analysis malware and stealthy malware still cannot survive dynamic analysis and reverse engineering, respectively. To provide an example, some researchers have proposed that data provenance analysis can effectively hunt stealthy malware [2].

Although the arms race between attackers and defenders has never stopped, the emergence of Gauss malware [11, 12] has made us realize that if attackers implement tailored malware for specific target systems, they can finally win this cat-and-mouse game. Additionally, the emergence of DeepLocker [13] makes us further realize that introducing new technologies (e.g., AI technology) can help attackers find greater advantages in building and applying such malware. To our knowledge, Gauss malware

* Corresponding author (email: cuix@zgclab.edu.cn)

Table 1 Comparison of three different malwares

Type	Description	Typical technique
Anti-analysis malware	Fight against static analysis, dynamic analysis, artificial reverse engineering, etc.	Encryption, confusion, polymorphism
Stealthy malware	Use various techniques to minimize their footprint among victims	Memory injection, macros and powershell
IUM	Hide the target and intention of the malware, so that it cannot be cracked.	Typical hash, DNN model

Table 2 Motivation cases for IUM

Type	Malware	Year	Key target attribute(s) of target victim(s)	Related technology
Cases in the wild	Gauss	2012	Specific configuration data	MD5 hash
	Equation (GrayFish loader)	2015	NTFS of system folder (%Windows% or %System%) Object_ID	SHA-256
	BIOLOAD	2019	COMPUTERNAME	CRC32 checksum, MurmurHash3
	APT41	2019	Specific user accounts, Volume Serial IDs	DPAPI (Data Protection API), RC5
	InvisiMole	2020	Local storage of credentials, such as login, Wi-Fi passwords	DPAPI
Cases in the lab	DeepLocker	2018	Face images, video, files, etc.	AlexNet Model

or DeepLocker still cannot be cracked. In the following analysis, we will note that this property is held because they meet the two core functions of "accurate identification" and "intention concealment". These two core functions are closely coupled and jointly determine an unbreakable security property. This security property carried by malware means that even advanced reverse methods cannot crack the hidden attack target and malicious intention. Therefore, we refer to this type of malware as "intention-unbreakable malware (IUM)".

Table 2 lists several motivative cases of IUM and their detailed descriptions can be found in Appendix A. Considering these cases and the related security analysis reports, we observe some key phenomena or information that is the core force driving our IUM threat research.

First, based on their analysis reports on these malware instances, major security companies have come to a more consistent conclusion that these malware programs protect their payload from security analysis-even if they find their components in telemetry or on malware-sharing platforms, they still cannot decrypt it outside the victim computer.

Second, from 2012 to 2020, such malware became more frequent in APT attacks.

Third, MITRE ATT&CK recorded most APT organizations and conducted TTP (tactics, techniques, and procedures) analysis of the malware or malicious activities used by these organizations, but gave no targeted defense recommendations in the TTP analysis of "environment keying" which is most closely related to IUM.

These phenomena constantly give us a dangerous impression that such malware is a new threat that could cause serious security problems, and no good defense or even defensive insights are found in the real world. To bridge this gap, this paper conducts the first modeling study on this kind of malware and analyzes questionable points that may be exposed to security researchers in the malware construction process through PoC implementation, which is expected to lay a key step for the subsequent defense research against such malware.

2 IUM modeling

This section first provides a formal model of IUM. The main goals of modeling include (1) performing risk quantification of existing and unknown IUM designs, (2) predicting new instances of IUM, and (3) boosting in developing the next-generation defense system to counter IUM-based attacks.

Table 3 Term description

Term	Type	Description	Ownership
iTotalSpace	Set	Feature space, the input space of IUM	iTotalSpace = iTrueSpace ∪ iFalseSpace
iTrueSpace	Set	Target feature space, characterize the attack target(s)	iTrueSpace $\subsetneqq$ iTotalSpace
iFalseSpace	Set	Non-target feature space, characterize the non-attack targets	iTrueSpace $\subsetneqq$ iTotalSpace
t_i	Instance	Target feature instance, e.g., a picture, a file	($t_i \in$ iTrueSpace) ∩ ($t_i \notin$ iFalseSpace)
f_i	Instance	Non-target feature instance	($f_i \notin$ iTrueSpace) ∩ ($f_i \in$ iFalseSpace)
cKey	Set	Candidate key space	cKey = {key, err}
key	Instance	Unique, len(key) $\geqslant$ 128 bit	key ∈ cKey, (key ≠ err) or (key ∉ err)
err	Instance/set	Uniqueness is not guaranteed, len(err) = len(key)	err ∈ cKey, (key ≠ err) or (key ∉ err)
Encryptor	Function	Encryption function	–
Decryptor	Function	Decryption function	–
iPlain	Program	Plain payload, executable program	iCipher = Encryptor(iPlain, key)
iCipher	Code	Cipher payload, non-executable binary code	iPlain = Decryptor(iCipher, key)
None	None	Means no output	None = Decryptor(iCipher, err)

2.1 Concept description

To facilitate the following description, this paper first defines and explains some commonly used terms, see Table 3 and the definitions in this section for details.

Malicious intention. Also known as attack intention, it is carried by the attack payload. Narrowly malicious intention usually refers to the attack behavior in which the attack payload is finally released, including intelligence theft, data encryption, hard disk erasure, data shredding, and so on. This paper focuses on the broader malicious intention, which not only refers to the attack behavior, but also refers to the attack target class and specific attack target instances of the attack payload.

2.2 Model definition

Definition 1 (IUM)**.** IUM is composed of a six-tuple, which reflects an executable software that is expected to achieve precise strikes on specific attack targets on the basis of ensuring the core functions of "accurate identification" and "intention concealment". The formal expression of IUM can be recorded as IUM = (iTotalSpace, iKeyGen, iCipher, iKeyDis, iBenCode, iProperty). The specific tuple description is as follows:

iTotalSpace. It is the input space of IUM and characterizes the size of the range in which IUM tries to find the attack target, reflecting the difficulty of locating the attack target. That is, the larger the scale of iTotalSpace, the less likely the attack target will be found.

iKeyGen. Candidate key generator. Its core functions are summarized as the following. (1) It can recognize the input without revealing the input information. (2) It can transform the input into a candidate key, as shown in the mapping matrix. Based on the mapping matrix, we define a series of evaluation indicators to measure the performance of iKeyGen in the following. (3) Candidate key deliverer, which directly delivers the candidate key to iKeyDis without exposing the information of the key.

iCipher. It is expressed as a non-executable binary code, also known as a cipher payload, which carries unbreakable malicious intention. It is the core objective of IUM to protect, and will eventually manifest itself as an attack behavior after precise strikes on the target host.

iKeyDis. Candidate key discriminator. Its core functions are summarized as (1) key judgment mechanism, which can judge whether the candidate key is the key or whether it can be used for decryption, without providing key-related knowledge; (2) perform decryption on iCipher without exposing the information of the key. Note: According to the different requirements of different application scenarios, the key judgment mechanism is also different and may not even be used, so it is recorded as iKeyDis = (keyJudger, Decryptor) or iKeyDis = (Decryptor).

iBenCode. iBenCode is short for benign code, which is a non-encrypted, executable, and benign application. IUM is a benign camouflage based on iBenCode.

iProperty. Unbreakable property. This is the inherent property of IUM. On the one hand, it can help IUM resist reverse engineering (advanced static analysis) by security experts; on the other hand, it can help IUM resist dynamic behavior analysis and prevent attack intentions from being exposed

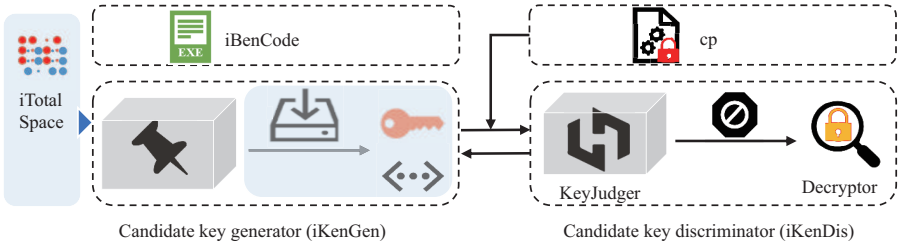


Figure 1 (Color online) Architecture of IUM model.

through attack behaviors. These can be summarized as two types of confrontation abilities, and we have established a systematic multi-dimensional analysis of them in Section 3 to help quantify the risk of IUM.

Above all, as shown in Figure 1, through the interaction between different components in the six-tuple, a hidden information flow of "input→key→payload" can be formed. This paper summarizes it as two main processes of "intention concealing" and "intention unlocking", which will be introduced in Subsection 2.3.

2.3 Model architecture

• **Intention concealing process.** The so-called intention concealing is simply to implement the encryption of the payload during the malware construction, and the key used to encrypt the payload is customized by the attacker based on the attack target, so that the encrypted payload is almost impossible to be detected before it reaches and attacks a pre-defined victim, and even if it is detected, its attack intention cannot be cracked. It should be emphasized that this process only occurs on the attacker's side, and the defenders are not aware of it.

Specifically, as shown in Algorithm 1, the process receives t and iPlain as input, and iCipher as output. First, iKeyGen realizes the function of transforming the target input into the key; then, the attacker uses Encryptor to accept the key as input, encrypts iPlain, and obtains iCipher, thus completing the intention concealing process. In the subsequent process, iCipher will be used as a core component for the construction of IUM instances. The construction process is not different from the construction of conventional malware, so this paper will not elaborate on it.

Algorithm 1 Malicious intention concealing process

Input: Target sample t specified by the attacker; iPlain;
Output: iCipher.
1: key ⇐ iKeyGen(t);
2: iCipher ⇐ Encryptor(iPlain, key);
3: **return** iCipher.

• **Intention unlocking process.** Corresponding to the intention concealing process, the intention unlocking process is to decrypt the cipher payload under certain conditions, otherwise, it will not decrypt. The specific condition here refers to if and only if the IUM delivered by the attacker achieves an accurate perception of the attack target. Specifically as shown in Algorithm 2, the process receives each feature instance (t or f) as input, and outputs iPlain or None.

When the input is the target feature instance (i.e., $i = t$), iKeyGen and iKeyDis obtain iPlain through operations such as accurately identifying the target, generating the key, transferring the key, authenticating the key, and decrypting the iCipher. After that, IUM releases the iPlain on the target host and performs precise strikes. When the input is a non-target feature instance (i.e., $i = f$), iKeyGen outputs err and passes it to iKeyDis. The key judgment mechanism (keyJudger) in iKeyDis will discriminate that err is not the key, and then output None. At this time, IUM keeps the intention hidden state on the current host.

Note: (a) As shown in Algorithm 2, m and n represent the size of the space of iTrueSpace and iFalseSpace that can be provided by the victim when the IUM is distributed to it, respectively. If the size of the feature space on the delivered host is fixed, the product of $(m + n)$ and the number of victims infected by the IUM reflect the range of delivery and distribution of the IUM by the attacker. (b) The limited input space on the target host will not affect the iProperty of IUM, which we will describe in detail through related indicators below.

Algorithm 2 Malicious intention unlocking process*

Input: The set of candidate samples for IUM, iVictimSpace* $= \{i_0, i_1, \ldots, i_{m+n-1}\} \in$ iTotalSpace;
Output: iPlain, if $i \in$ iTrueSpace;
None, if $i \in$ iFalseSpace.

```
while q ⩽ (m + n − 1) do
    if i_q ∈ iTrueSpace then
        key ⇐ iKeyGen(i_q);
        iPlain ⇐ iKeyDis(iCipher, key);
        return iPlain;
    end if
    if i_q ∈ iFalseSpace then
        err ⇐ iKeyGen(i_q);
        None ⇐ iKeyDis(iCipher, err);
    end if
end while
```

* iVictimSpace refers to the limited input space that can be provided by the victim when the IUM is distributed to it.

2.4 Attack scenarios

The realistic assessment of the new security threats represented by IUM relies on the clear attack scenario. To this end, referring to the authoritative cyber threat frameworks [14–18] and the analysis of well-known cybersecurity incidents, this paper provides the threat attack chain for IUM, and describes the attack scenarios applicable to IUM.

The attack chain consists of 7 stages, namely reconnaissance, construction, delivery, persistence, lateral movement, evasion, and impact.

(1) Reconnaissance. In this stage, the attacker will try to identify high-value attack targets, and collect target victim information through active scanning, phishing, social engineering, etc., or the attacker can implant identifiers into the target host. The information carried by these identifiers can be specific system configuration information, specific target files, specific character pictures, or specific audio, video, geographic location, etc. They are only associated with specific target hosts or a specific target group, and are used by IUM to accurately identify attack targets.

(2) Construction. According to the model architecture shown in Figure 1, the construction of IUM can choose different technologies to construct iKeyGen according to the type of information associated with the attack target, such as the value transfer black box, hash black box, and AI black box. We will introduce these technologies in detail below; corresponding to the construction of iKeyDis, this paper uses the AES128 or AES256 algorithm by default for the implementation of Encryptor and Decryptor. Among them, keyJudger also has a variety of different implementation technologies. In theory, in addition to the value transfer black box, all other techniques used to construct iKeyGen can be used to implement keyJudger.

Taking supply chain pollution [19–22] as an example, iBenCode can be selected as a commonly used application for a specific attack target (group), a video conferencing application (represented by programA) is an example of iBenCode, then IUM is the programA with iKeyGen, iKeyDis, and iCipher embedded. Finally, IUM is a kind of payload-agnostic malware. Under the premise that the volume increment of IUM relative to iBenCode is negligible, iCipher can be the encrypted result of any kind of executable payload.

(3) Delivery. The delivery of IUM to the victim host includes but is not limited to the following forms: (a) supply chain pollution; (b) widely and randomly distributing IUM; (c) spreading itself by IUM throughout the network. A real-world example is where malware utilizes lateral movement to identify a physically isolated private network or to locate high-value attack targets in an organization.

(4) Persistence. IUM implements a precise strike function that only targets the target host(s), making it completely opposite to the behavior displayed on non-target hosts (i.e., intention concealment).

(5) Lateral movement. Through software sharing or IUM's own lateral movement to establish the persistence on other machines, so as to finally locate the attack target.

(6) Evasion. This paper assumes by default that the attack target has strong defensive capabilities, but when the IUM does not reach the target host, due to the lack of target feature instances, even the use of advanced reverse analysis techniques cannot crack the malicious intention of IUM. For this, we will perform analysis and evaluation in the following.

(7) Impact. When IUM finally arrives at the target host, because the required specific conditions have been met, the intention unlocking process will be triggered and executed immediately, and finally IUM decrypts and executes the attack payload and releases the attack behavior.

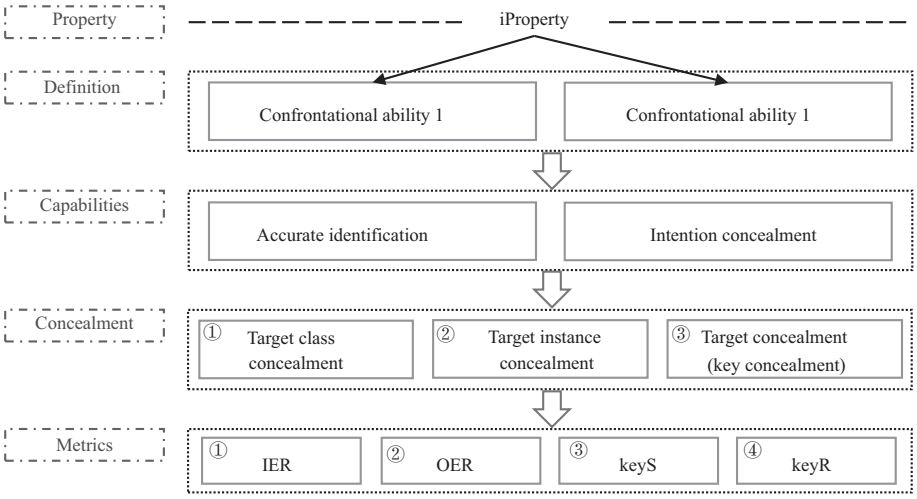


Figure 2 Multi-dimensional description of iProperty.

It is worth mentioning that there is no C&C channel in the attack based on IUM. IUM belongs to one-time (static) updated malware and acts alone on the target victim, so IUM does not rely on domains and IP addresses related to malicious activities. That is, IUM can use authoritative domain names (or websites based on authoritative domain names) to distribute or spread itself to victim hosts, which can bypass the detection based on the domain reputation system [23]. In addition, if the infected machine is not directly accessible (for example, they are in a private home), and the scale of the infection is large, it is not possible to implement repair strategies such as deleting the malware, restoring data, and services from secure backups. More importantly, the defender cannot protect and trace the source by canceling the C&C channel, which once again highlights the new threats represented by IUM.

In addition to the advantages brought by the absence of C&C channels, other advantages of IUM threat attacks can be summarized as follows.

(1) The anti-virus (AV) solution, or intrusion detection system (IDS) defense scheme can be bypassed, because IUM does not show any abnormality in the detection of network data packets before reaching the target host.

(2) It can resist static analysis. Typical static features include instruction features, control flow graphs, and byte value patterns [24]. However, these features can at most allow us to discover/detect the existence of the code logic that "the encrypted binary code is decrypted and triggered to execute", but how to decrypt and trigger the execution of the binary code depends on the key that is dynamically generated by iKeyGen and cannot be reversed or brute-forced (see the analysis below for details). Therefore, the encrypted binary code will not be judged as malicious by AV or IDS, or other malware detectors.

(3) It can resist dynamic analysis by embedding an unknown payload. The component payload can integrate new or unknown behavior patterns so that the corresponding malicious activities can evade the detection of AV and IDS; and because the defender cannot achieve the decryption of the unknown cipher payload, a type of attack mode implemented by the payload can be reused by attackers and can guarantee the effectiveness of the attack, thus re-emphasizing the importance of cracking the attack intention and achieving defense from the root cause. Note: The payload research is orthogonal to the IUM modeling research, which is beyond the scope of this paper. We consider conducting in-depth research on payload construction in the future.

3 Security property analysis

In addition to the above qualitative analysis in the IUM modeling process, as shown in Figure 2, this paper establishes a top-down multi-dimensional interpretation around iProperty and quantifies the unbreakable security capabilities of IUM through a set of indicators.

Table 4 Mapping matrix of IUM[a)]

Mapping matrix		Real sample	
		t	f
Mapping result	key	TK	FK
	err	TE	FE

a) T: True, F: False; K: key, E: err.
TK: IUM obtains the key by accurate identification when its input is t.
TE: IUM outputs err when it receives the input t, which means a false negative due to its inaccurate identification.
FK: IUM obtains the key when it receives the input f, which means a false positive due to its inaccurate identification.
FE: IUM outputs err by accurate identification when its input is f.

3.1 Two core capabilities and three-layer concealments

The two confrontational capabilities of IUM are specifically manifested as accurate identification and intention concealment. They jointly determine the iProperty of IUM by implementing three-layer concealments.

Accurate identification. This capability helps IUM to perceive whether the current host is the target victim without revealing the relevant knowledge of the target feature, thus realizing two layers of concealment, namely, target class concealment and target instance concealment.

• Target class concealment means that the defender cannot discover the type of target that an IUM is looking for (e.g., person or organization).

• Target instance concealment means that even if the target class is known, the defender cannot discover the specific object that the IUM is looking for (e.g., the facial image of Tom Cruise or a specific network environment).

Intention concealment. This capability conceals the effective attack payload (i.e., iPlain). Specifically, IUM hides the malicious behavior by fully encrypting iPlain to iCipher. Therefore, the release of attack intention depends on the unlocking of iCipher. The unique key used to unlock iCipher is dynamically generated by iKeyGen and is not hard-coded by IUM. Therefore, intention concealment means that the defender cannot obtain the unique key through advanced analytical techniques.

In summary, IUM realizes the three-layer concealment of the target class, the target instance, and the malicious intention. When IUM is released, since it does not carry any plaintext information about the attack target and attack intention, IUM itself does not know its attack target and attack intention. If and only when the released IUM reaches the target host, the iKeyGen model in IUM will generate a key based on the features of the target host and use the iKeyDis model to decrypt the cipher payload.

In addition, according to the introduction below, iKeyDis (designed to) has natural black box characteristics II. That is, iPlain is the output of iCipher and key based on a certain functional relationship, and the iPlain-related knowledge cannot be provided by iCipher and key. Therefore, assuming that the iCipher is captured by the attacker, intention concealment is equivalent to that the defender cannot obtain the unique key through advanced analysis techniques.

3.2 A set of metrics

According to the above analysis, the most directly related objects of IUM's three-layer concealments are the attack target and the key. Therefore, the quantitative index of iProperty is proposed around these two objects. In this subsection, we use the mapping matrix (shown in Table 4) to support the definition of the following quantitative indicators.

Input enumeration rate (IER). IER represents the number of target samples that can be mapped to the key in the entire input space, reflecting the possibility of achieving intention unlocking in the input space. A lower IER corresponds to a lower probability of unlocking IUM through forwarding enumeration. From the attacker's point of view, IER should be infinitely close to or equal to 0. Specifically, the function of IER is defined as

$$\mathrm{IER} = \frac{\mathrm{TK}}{\mathrm{TK} + \mathrm{FK} + \mathrm{FE} + \mathrm{TE}}. \tag{1}$$

Output enumeration rate (OER). OER is expressed as the probability of enumerating the unique key from the entire output space, reflecting the possibility of unlocking the intention from the output space. Specifically, the function of OER is defined as

$$\mathrm{OER} = \frac{1}{2^{\mathrm{len(key)}}}. \tag{2}$$

Among them, len(key) represents the binary length of the key; from an attacker's point of view, OER is expected to be infinitely close to or equal to 0.

key stability (keyS). keyS is expressed as the statistical probability of all real target samples that can be mapped to the key, which reflects the stability of key generation. keyS also reflects the ability of IUM to perform the precise attack on the target victim, that is, the higher the keyS, the higher the precise attack rate. From the attacker's point of view, keyS should be infinitely close to or equal to 1. Specifically, the function of keyS can be defined as

$$\text{keyS} = \frac{\text{TK}}{\text{TK} + \text{FE}}. \tag{3}$$

key reachability (keyR). keyR represents the probability statistics of real non-target samples successfully generating the key through collisions, reflecting the possibility to brute force the attack intention. Therefore, keyR characterizes the ability to hide intention. The lower the keyR, the stronger the intention hiding ability. From the attacker's point of view, keyR should be infinitely close to or equal to 0. Specifically, the function of keyR is defined as

$$\text{keyR} = \frac{\text{FK}}{\text{TE} + \text{FK}}. \tag{4}$$

In summary, the ideal unbreakable property requires the following: (1) IER, OER, and keyR infinitely approach 0; (2) keyS infinitely approaches 1. Based on these four sets of indicators, on the one hand, it can help IUM's PoC design and implementation; on the other hand, it can also allow us to make more meaningful risk assessments of IUM instances in the past or in the future.

It is worth mentioning the following.

(1) Referring to the security standards defined in cryptography, 2^{128} is the critical value that security experts cannot achieve brute force unlocking. Based on this value, we define a space that can be enumerated at least 2^{128} times as an infinite space. Therefore, we require that the input space iTotalSpace of IUM and the enumeration space of key should be at least 2^{128}, that is TK + FK + FE + TE $\geqslant 2^{128}$; len(key) $\geqslant 128$.

(2) In addition, the attack target of IUM is usually a unique or limited group, so the target feature space represented by TK + FE is limited.

(3) The selection of target features (or input space) is very important for the construction of IUM. Good target features will be hard to spot and produce high entropy, unguessable inputs. Therefore, a comprehensive analysis of target features in terms of concealment and entropy will be very necessary. However, it is independent of the research scope of this paper, and the author will take it as another subject to carry out research in the future.

4 Taxonomy and predictive studies on IUM

According to the IUM model architecture, in the design and implementation of IUM's PoC, iKeyGen and iKeyDis are the key components used to ensure iProperty. In order to meet the iProperty security property of IUM, in addition to following the above four sets of indicators, we have carried out a black box design for iKeyGen and iKeyDis. Specifically, this paper defines two black-box characteristics (BBCs) to support the taxonomy study on IUM, which provides a theoretical basis for summarizing real-world attack cases, predicting new IUM, and exploring new extended features (e.g., generalization consistency) of IUM.

4.1 Two black-box characteristics

Black box characteristics I (BBC-I). Let H denote the black box model, t denote its input and key denote its output. Then the black box characteristics are summarized as (1) the knowledge of t is unknowable; (2) the knowledge of the key is unknowable; (3) the key is the output of t based on a certain functional relationship. On the basis of satisfying the characteristics of (1) and (2), whether the relationship is linear, or transparent, or whether it provides mutual knowledge is not cared about.

Black box characteristics II (BBC-II). Let H denote the black box model, the key denotes its input, and M denote its output. M is the output of the key based on a certain functional relationship,

and the black box characteristics are summarized as (1) the knowledge of the key is unknowable; (2) the knowledge of M does not provide the knowledge of the key.

Note: The above black box characteristics are defined from the perspective of the defender, so "knowledge is unknowable" is effective for the defender but invalid for the attacker.

Based on the proposed two types of BBCs, this paper conducts taxonomy and predictive studies on IUM.

First, inspired by the existing attack cases (Table 2) and related research work, this paper summarizes three types of algorithms that meet BBC(s), namely, the value transfer algorithm, typical hash algorithm, and DNN algorithm, and corresponding model (IUM) is called (1) value transfer black box (*-based IUM), (2) typical hash black box (*-based IUM), (3) deep neural network (DNN) black box (*-based IUM).

Second, since DNN models tend to have large sizes, there have been challenges in building practical application-oriented DNN black box-based IUM. It is important to conduct a further thorough analysis of such threats so that researchers can have insights into the degree of threat that IUM may pose in the future. In view of this, this paper deeply examines the limitations or extensibility of IUM threats around the substitutability and practical feasibility of DNN technology. Specifically, in terms of substitutability, this paper proposes a new type of IUM based on a perceptual hash black box ((4)). In terms of practical feasibility, this paper breaks through the volume limit of the DNN model in building IUM, and realizes the small volume increment from laboratory to practical application.

From the perspective of classification and prediction research, four types of black boxes for building IUM are introduced below. Specific implementation schemes of these black boxes will also be provided in the experimental section. Among them, iKeyGen is designed following BBC-I, and iKeyDis following BBC-II. In addition, the AES encryption algorithm used by Decryptor naturally fits BBC-II, so only keyJudger needs to be considered when designing iKeyDis.

4.2 BBCs-based taxonomy and prediction scheme

Here are four BBC-based schemes we concluded in terms of taxonomy and prediction. Appendix B introduced the research related to these schemes for supporting our taxonomy and prediction.

• **Value transfer black box.** The value transfer black box is implemented based on the value transfer algorithm. The black box model is constructed mainly through operations such as combination, splicing, cropping, and assignment. This type of black box is the simplest and most direct way to map t to the key. Value transfer black box generally takes character strings as input and can adapt to different feature spaces. For example, the victim's physical environment, software environment, and geographic location often use character strings to describe themselves. To ensure that the feature space is unlimited, the string length corresponding to the target feature instance specified by the attacker should be at least 16 bytes (128 bits), and the 16 bytes as the input source of the black box model should be valid 16 bytes. That is, unstructured data cannot be characterized, and there are no restrictions or requirements for specific characters.

• **Typical hash black box.** Typical hash algorithms such as sha256 and sha512 are recognized by security experts as meeting the two attributes of "anti-collision" and "irreversibility", and can naturally be used to build black box models. This type of black box generally receives files or character strings as inputs and its adaptable input space includes the input spaces covered by the value transfer black box and the input space represented by massive files.

• **Deep neural network black box.** A DNN with a good classification or prediction function can complete the black box construction through its hidden layer or multi-class output. The hidden layer or the multi-class output layer realizes the dynamic concealing of the key and the mapping from a class of target feature instances to the unique key. This type of black box generally receives a text or picture as input, and its adaptable input space includes the input space covered by the typical hash algorithm, which is characterized by massive network pictures.

• **Perceptual hash black box.** The perceptual hash black box is a black box design scheme proposed by this paper to assist the construction of IUM for the first time. It is implemented based on the perceptual hash algorithm, and similar to DNN black box, it also implements an "one class-to-one" mapping relationship. A perceptual hash algorithm generally receives a picture as input and can generate the same "fingerprint string" for a class of very similar pictures. As shown in Figure 3, the difference between these pictures is only reflected in the scattered pixel values, and these pictures are very similar or even indistinguishable in visual perception. When any picture in the network resource is designated

(a) (b)

Figure 3 (Color online) Target class pictures for perceptual hash black-box. (a) t_0: initial target picture; (b) t_i: target picture (t_i is not t_0).

as its target input, a new class of target inputs can be generated by modifying its pixel value, and they are mapped to the same expected output by perceiving the hash black box. However, all other massive pictures in the network resources constitute its non-target input space, and they are mapped to various unexpected outputs by the perceptual hash black box. This process greatly facilitates collaborative adversarial attacks, and can cleverly avoid signature or hash-based detection, thereby ensuring that other victims will remain hidden when a victim is detected. There are five types of perceptual hash algorithms, namely, aHash, pHash, dHash, wHash, and colorHash [25,26], which can support the construction of this type of black box model. In turn, many different IUM instances can be constructed.

• **Brief summary.** Let Value_Transfer, Typical_Hash, DNN_Bucket, and Perceptual_Hash denote the implementation functions of the above four types of black boxes in turn, with N as its input and Y as its output. Then they can be defined in simple formalization, as shown in Table 4 for details.

Comparing the two types of black box characteristics, it is not difficult to conclude: In the realization of the value transfer black box, the knowledge of Y will provide the knowledge of N. Therefore, as shown in Table 5, the value transfer black box can only satisfy BBC-I, while the other three types of black boxes can all meet BBC-I and BBC-II at the same time.

In summary, iKeyGen has four different implementation schemes, and iKeyDis has three different implementation schemes. However, according to the model architecture of IUM, the input of iKeyDis is the output of iKeyGen. Since the output of iKeyGen is unique, according to the principle of transitivity, the input of iKeyDis will only be unique. Although the DNN black box and the perceptual hash black box can be used to implement iKeyDis (i.e., simplify the "one class-to-one" mapping to the "one-to-one" mapping), they are not necessary. Therefore, this paper only uses the typical hash black box scheme in the implementation of iKeyDis.

4.3 Taxonomy study based on generalization consistency

The exploration and prediction on the perceptual hash-based IUM bring new insights for us. That is, on the basis of meeting iProperty, the implementation of IUM does not limit the unique deterministic input to identify attack targets, and the DNN model is not the only technical solution to achieve accurate target identification based on a class of inputs.

From this conclusion, this paper further defines another optional but important function — generalization consistency. Generalization consistency refers to the completely consistent output of the same class of target features (used to identify the attack target), which is strictly different from the output corresponding to the non-target features.

For ease of description, we refer to the IUM without generalization consistency as the basic IUM, or otherwise, as the generalized consensus IUM. The specific classification is shown in Table 5.

Basic IUM can guarantee attackers can generate different environment keys based on different targets. These keys can be leveraged by Encryptor to obtain a batch of different cipher payloads, which can realize the continuous exploitation of an attack intention, and effectively reduce the attack cost, but for defenders and ordinary users, will bring great security threats.

The generalized consensus IUM further expands the attack scene and even directly achieves adaptation to unknown attack scenes through generalization ability to unknown target samples. Taking the face images of a specific character as the target attributes as an example, theoretically, no matter what background the target character is in, DeepLocker can accurately capture the target face images in different backgrounds with the AI model.

5 Experiment

5.1 Implementation of iKeyGen

• **Value transfer black box-based iKeyGen.** We locate the target victim in a network environment by identifying the target service set identifier (SSID) and the target basic SSID (BSSID). SSID is unstructured data that can be set at any length within the (0, 256] bit zone. Considering the influence of the change of SSID length on the security strength of the key, this paper uses the combined information of tSSID and tBSSID to generate the key as specified in (5).

$$\text{key} = \begin{cases} \text{key}_{\text{len(tSSID)}}(\text{tSSID}) + \text{key}_{\text{128-len(tSSID)}}(\text{tBSSID}), & \text{(i)}, \\ \text{key}_{128}(\text{tSSID}), & \text{(ii)}, \end{cases} \tag{5}$$

where,

(1) tSSID is the target SSID and tBSSID is the target BSSID, both of which are designated by the attacker after preliminary reconnaissance;

(2) len(tSSID) refers to the length of tSSID;

(3) $\text{key}_l(y)$ is a part of the key generated by y, and its length is l;

(4) "+" denotes "combination" or "splicing".

Specifically, Eq. (5) provides two different schemes to generate keys, depending on the length of tSSID.

When the length of tSSID $\leqslant$ 128 bits, we would use the complete binary representation of tSSID to form part of the key, and the remaining part of key would be complemented by tGUID. In this case, the final 128-bit key is a combination of tGUID and tSSID information.

When the length of tSSID $>$ 128 bits, we perform a clipping operation on tSSID and extract the content of 128 bits length to generate the key.

Value transfer black box-based iKeyGen implements a "one-to-one" function mapping relationship. Therefore, in the measurement of iProperty, (1) the conclusion of TK = 1, FE = 0 and FK = 0 is clear; (2) according to our settings, the sizes of the input and output spaces are both 2^{128}, therefore, TE = $2^{128} - 1$. Furthermore, the four metrics of iProperty are calculated as follows IER = $1/2^{128}$, OER = $1/2^{128}$, keyS = 1, keyR = 0.

• **Typical hash black box-based iKeyGen.** We use a specific target file to locate the target victim and construct the iKeyGen based on the sha256 algorithm as shown in (6):

$$\text{key} = \text{sha256}(\text{tFile}), \tag{6}$$

where tFile refers to the target file designated by the attacker; the generated key is 256 bits in length; and the default assumption is that the length of the text contained in tFile should be no less than 128 bits.

Similarly, the typical hash black box realizes a "one-to-one" mapping relationship. Therefore, the conclusion that TK = 1, FE = 0 and FK = 0 is clear. In our implementation, given that our specified target file is unique, the non-target file space is the set of all network files except for the target file, which is massive and non-enumerable. Therefore, TE $\gg 2^{128} - 1$. Moreover, given that len(key) = 256, the metrics of iProperty are calculated as follows: IER $\ll 1/2^{128}$, OER = $1/2^{256}$, keyS = 1, keyR = 0.

• **Deep neural network black box-based iKeyGen.** We use facial images of Tom Cruise (Figure 4) and the binary classification DNN (B-DNN, its unfolded form is as shown in Figure 5) to locate target victims and construct iKeyGen based on the B-DNN black box as shown in Figure 5.

Specifically, we have the following.

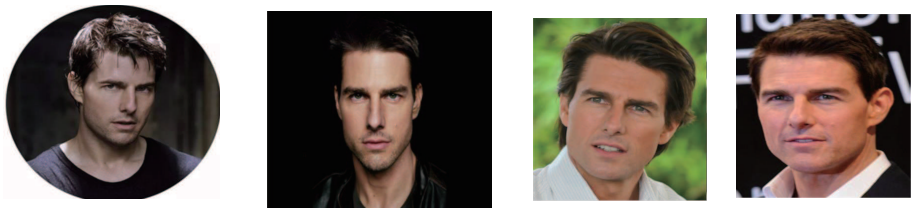
(1) Conv represents the convolutional neural network structure composed of convolutional layers, activation layers, and pooling layers.

(2) Affine represents the fully connected neural network structure composed of fully connected layers, activation layers, and dropout layers.

(3) AffineK represents a fully connected layer with K neurons. The second Affine128 layer shown in Figure 5 is the layer that we specify to determine the output of the iKeyGen.

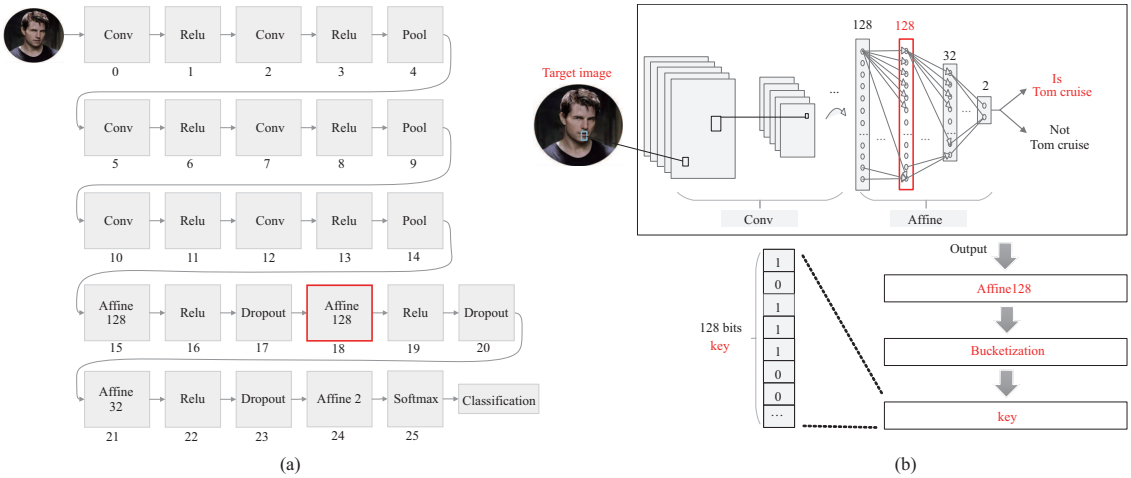
(4) Bucketization represents a generalization mechanism that divides the 128 output values of the hidden layer of the model into two buckets representing 0 and 1, to ensure the stability of the mapping from the same type of target to the unique key.

The B-DNN black box-based iKeyGen implements a "one class-to-one" mapping relationship. In particular, it is impractical to train B-DNN with massive natural images, so this paper adds adversarial



(a) (b) (c) (d)

Figure 4 (Color online) Target class pictures for AI black-box. (a) Example 1; (b) example 2; (c) example 3; (d) example 4.



(a) (b)

Figure 5 (Color online) B-DNN model. (a) Unfolded layers of B-DNN; (b) B-DNN black box-based ikeyGen.

Table 5 A brief summary of the four types of black boxes

Black box function	BBC-I	BBC-II	Accurate identification	Intention concealment	Generalization consistency	Target input	Target output	Description
$Y = \text{Value\_Transfer}(N)$	✓	×	✓	✓	×	Unique	Unique	The simplest implementation is $Y = N$
$Y = \text{Typical\_Hash}(N)$	✓	✓	✓	✓	×	Unique	Unique	The simplest implementation is $Y = \text{sha256}(N)$
$Y = \text{DNN\_Bucket}(N)$	✓	✓	✓	✓	✓	Not unique	Unique	As shown in Figure 5
$Y = \text{Perceptual\_Hash}(N)$	✓	✓	✓	✓	✓	Not unique	Unique	As shown in Figure 6

training to B-DNN, which is a measure commonly adopted by current neural networks to enhance the classification effect for unknown data. Specifically, we construct our dataset by crawling specific face images based on the Google images [27], and find that higher quality and a larger number of face images can be obtained than the existing well-known datasets [28–34], as shown in Table 6. Then, our dataset is split into the labeled dataset (target sample: 1317, non-target sample: 1757) and the unlabeled dataset (target sample: 673, non-target sample: 786). The labeled dataset is used to learn the expression and classification of features, and the unlabeled dataset is used to assist data to enhance the feature expression. In view of the fact that DNN is generated based on limited input space training, the metric values of keyS and keyR are obtained through approximate calculations: keyS = 99.9%, keyR = 0, where TK = 1989, FE = 1, FK = 0, TE = 2543.

This result brings great confidence to the actual deployment of IUM based on the AI black box.

IER and OER are calculated based on the actual deployment scenario. Specifically, this paper obtained a total of 65000 non-target images from Google images [27] and VGG Face2 [30], which were used to test IUM. The test results show that the B-DNN black box-based IUM fails to release the attack succeeded

Table 6 Comparison of different face datasets

	Celeba [28]	FaceScrub [29]	VGGFace2 [30]	BIWI [31]	YouTube [32]	PubFig [33]	LFW [34]	This paper
Pics/person	20	200	362	750	97.5	294	2.3	**1000+**
Identify	Facial feature	Gender	Posture and age	Posture	Person	Person	Facial feature	Person

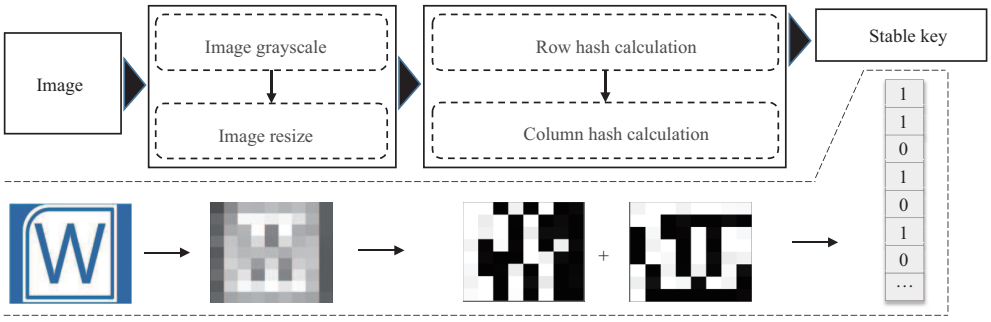


Figure 6 (Color online) Perceptual hash black box-based iKeyGen.

because the facial images of the target person (Tom Cruise) are not included in the two datasets, thereby guaranteeing that the IER is small enough to a certain extent. Moreover, len(key) = 128. Therefore, IER and OER are calculated as IER = O(target class)/O(total classes) $\ll 1/2^{128}$, OER = $1/2^{128}$.

• **Perceptual hash black box-based iKeyGen.** We use the slightly modified and visually indistinguishable images of the Microsoft Word application (as shown in Figure 3) to locate the target victim(s). This is very consistent with the attack scenario associated with the distribution of malicious documents. For example, malware families such as Emotet [35], QakBot, and Ursnif [36] bypass malicious activity tracking based on hash calculations by generating variants of malicious document images [37]. This paper shows the construction of the black box model based on dHash, as shown in Figure 6.

The construction process of this black box can be summarized in the following steps.

(1) Given an input image (we take the target image as an example), convert the image to grayscale, that is, Gray ← RGB.

(2) Reduce the image size to 9×9.

(3) Calculate "row hash" through the perceptual hash function to obtain the 64-bit row hash value.

(4) Calculate "column hash" through the perceptual hash function to obtain the 64-bit column hash value.

(5) Concatenate the row hash and column hash values to synthesize a 128-bit hash value and output this value as key.

For the perceptual hash black box-based iKeyGen, a class of target feature instances it uses is created and controlled by the attacker. Except for the initial target image originally specified from the network, the other target feature instances are not part of the network resource before being delivered by the attacker. Therefore, in terms of the input space that the defender can enumerate, the "one class-to-one" mapping relationship implemented by the perceptual hash remains essentially equivalent to the "one-to-one" mapping relationship.

Furthermore, when implementing this type of black box, TK = 1, FE = 0, FK = 0, TE $\gg 2^{128}$, and len(key) = 128 are all clear. In this case, the metrics of iProperty are calculated as IER $\ll 1/2^{128}$, OER = $1/2^{128}$, keyS = 1, keyR = 0.

5.2 Implementation of iKeyDis

In the PoC implementations provided in this paper, two different iKeyDis implementation schemes are provided.

(1) iKeyDis = (keyJudger, Decryptor), where keyJudger is the implementation of a typical hash black box. The specific pseudo code is shown in Algorithm 3.

Among them, M is the hash of the key, which is the plaintext information embedded in the IUM.

(2) iKeyDis = (Decryptor), that is, directly use Decryptor to try to decrypt iCipher. The specific pseudo code is shown in Algorithm 4.

Comparing the two types of implementation schemes, the former brings a more obvious advantage through the integrated key judgment mechanism, that is, it can reduce the probability of IUM being

Algorithm 3 iKeyDis implementation scheme1

Input: ckey; iCipher;
Output: abs, or, null;
1: **if** Typical_Hash(key) == M **then**
2: iPlain ⇐ Decryptor(iCipher, ckey);
3: abs ⇐ execute(iPlain);
4: **return** abs;
5: **else**
6: **return** null;
7: **end if**

Algorithm 4 iKeyDis implementation scheme2

Input: ckey; iCipher;
Output: abs, or, null;
1: iPlain ⇐ Decryptor(iCipher, ckey);
2: abs ⇐ execute(iPlain); `// abs=0 means ‘‘execution success", else means ‘‘execution failed"`
3: **if** abs == 0 **then**
4: **return** abs;
5: **else**
6: **return** null;
7: **end if**

Table 7 The six-tuple composition and VI of three IUMs

IUM = {iTotalSpace, iKeyGen, iCipher, iKeyDis, iBenCode, iProperty[a)]} ⇒ VI						
No.	iTotalSpace	iKeyGen	iCipher	iKeyDis[b)]	iBenCode	VI (MB)
1	iTrueSpace = $\{t\}$ = {unique tSSID} iFalseSpace = $\{\forall \text{string} \neq t\}$	Value transfer black box	The encrypted output of a.exe	AES-128	A simple pop-out program (20.24 MB)	0.99
2	iTrueSpace = $\{t\}$ = {unique tFile} iFalseSpace = $\{\forall \text{file} \neq t\}$	Typical hash black box		AES-256		0.99
3	iTrueSpace = $\{t_0, t_1, \ldots, t_m\}$ = {Tom Cruise's facial images} iFalseSpace = $\{\forall \text{Images} \neq t_i \vert (i = 0, 1, \ldots, m)\}$	B-DNN black box		AES-128		4.27
4	iTrueSpace = $\{t_0, t_1, \ldots, t_m\}$ = {Microsoft Word's images} iFalseSpace = $\{\forall \text{Images} \neq t_i \vert (i = 0, 1, \ldots, m)\}$	Perceptual hash black box		AES-128		2.46

a) iProperty is an inherent property of IUM, which has no actual visibility or volume.
b) This paper uses the iKeyDis = (keyJudger, Decryptor) scheme by default, in which keyJudger is implemented using a typical hash black box scheme.

Table 8 Source of IUM's VI

IUM No.* \ VI		Code (kB)	Non-code (kB)	iCipher (kB)
Non-NN method	No. 1	21740–20723	0	452
	No. 2	21738–20723	0	
	No. 3	23245–20723	0	
NN method	No. 4	23247–20723	NN: 1850	

detected as suspicious malware by reducing frequent attempts to decrypt operations. Of course, depending on the different attack scenarios, the code with hash judgment may also become a high-entropy suspicious code; then the second scheme may have more advantages.

5.3 Four PoC implementations of IUM

Based on the design and implementation of the above-mentioned iKeyGen and iKeyDis, four types of IUMs are finally constructed. The tuple composition of IUM is shown in Table 7. Last but not least, in order to minimize the possibility of being suspected during the attack, the volume increment (VI) of IUM (relative to iBenCode) was also token as an important factor in determining whether IUM can be deployed as real malware.

As shown in Table 7, iTotalSpace, as the input of IUM, does not bring VI to IUM. Therefore, the main contributors to the VI of IUM are iKeyGen, iKeyDis, and iCipher. According to its implementation, we divide IUM into two categories, one is the IUM constructed based on the neural network (NN) method, whereas the other is the IUM constructed based on the non-NN method. The details are shown in Table 8.

Table 9 Some selected payload for constructing IUM

Name	Stuxnet	ZeusVM	Destover	Asprox	Bladabindi
Size (MB)	0.02	0.05	0.08	0.09	0.10
Name	EquationDrug	Kovter	trickbot	Cerber	Ardamax
Size (MB)	0.36	0.41	0.44	0.59	0.77

Table 10 Detection results of antivirus engines

Antivirus engine		iPlain	iCipher	IUM
Host	Kaspersky	Kill immediately	0 risk	0 risk
	Windows defender	Kill immediately	0 risk	0 risk
Cloud	VirusTotal	57/72	0/61	Value transfer black box-based IUM: 1/72
				Typical hash black box-based IUM: 1/71
				Deep neural network black box-based IUM: 1/70
				Perceptual hash black box-based IUM: 1/71
	ThreatBook	9/25	0/25	–

The latter category is a form of "code is implementation". This VI comes from the building code of iKeyGen and iKeyDis (including the dynamic link library). We refer to this VI as "code VI".

For the former category, in addition to the "code VI", the construction of iKeyGen also relies on training the neural network and saving the model structure and parameters, which will bring another volume increase. We call this VI "non-code VI".

iCipher (or iPlain) brings the third type of VI. Due to the difference in the type, function, or behavior of the payload, the volume of the payload also differs. We capture nearly 4000 payload samples that are labeled "malicious" from public cloud sandboxes [38,39] and theZoo aka Malware DB [40]. The statistical result reports that almost half of the payloads are less than 1 MB (Table 9 shows part of the available payloads used for IUM batch instantiation). Compared with that of BenCode, the volume of the payload is negligible. As stated in Table 5, in this paper, we choose a.exe in trickbot as payload, whose volume stays at 452 kB before and after encryption.

As shown in Tables 7 and 8, the VI range interval we implement is [0.99 MB, 4.27 MB]. From the VI metric perspective, a smaller VI and larger BenCode volume correspond to a lower likelihood for IUM being detected as suspicious. Therefore, in actual attack scenarios, IUM construction needs to consider an acceptable VI ratio in order to select an appropriate iBenCode for integration.

Above all, the ability of IUM to resist static analysis has been guaranteed by iProperty's four indicators in the process of design and implementation. In addition, as shown in Table 10, in order to verify its ability to counter dynamic analysis, this paper uses two methods: cloud sandbox detection and secure host detection. First, we submit all IUM instances to 72 cloud antivirus engines [38,39,41], and IUM has achieved zero risk avoidance for all these instances. Second, in the host-based test, we chose Kaspersky and Windows Defender as the two AV engines, and IUM also achieves almost zero risks of being detected and killed, further confirming the emergence of new security threats brought by IUM. Thereby, it is a strong proof that IUM can create avenues for adversaries to counter the defense mechanisms, especially NN-based IUM, which is a promising malware in real-world applications. In the future, we will conduct in-depth investigations on more exploit instances by deducing the practical threat scenarios. There are some possible directions including (1) automatic distribution and delivery of IUMs in a non-cooperative environment, (2) coordinated botnets based on IUM, which can support autonomous decision-making and fault-tolerant attacks in an uncontrolled environment, and (3) attack effectiveness evaluation of IUMs in offensive and defensive cyber ranges.

6 Mitigation measures

Totaly, we provide insights from 7 aspects into how to counter IUM. However, due to the limitation of the paper length, here we only focus on two among them, and others are introduced in Appendix C.

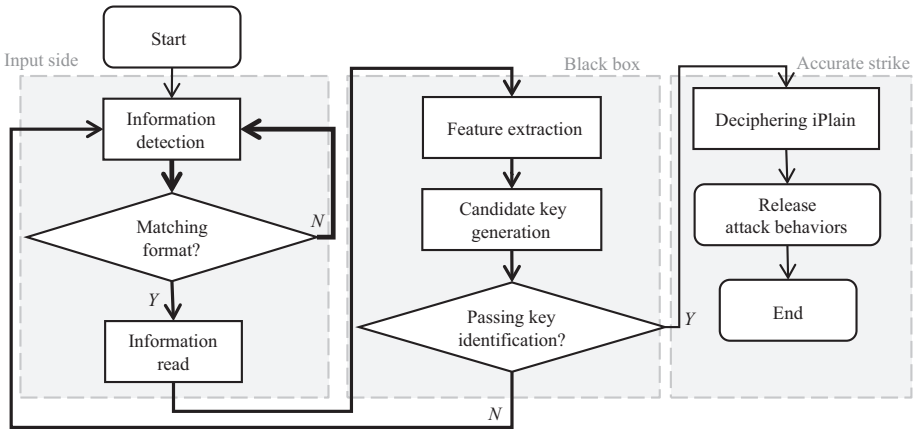


Figure 7 Behavior sequence of the IUM (Note: the thicker the connection line, the more repetitions).

6.1 The behavioral sequence-based graph detection method

According to the aforementioned threat analysis of IUM, its attack features can be summarized as follows.

(1) In order to ensure the secrecy of the target image in the input space, the delivery of IUM should be distributed arbitrarily, and the process of accurate target positioning will involve a large number of (non-target) information detection, feature extraction, and other operations.

(2) Feature extraction, candidate key generation, and key identification based on the black boxes are the decisive process of obtaining the unique environment key. However, it is rare to see the activity chain of using the output of the black-box model for attempted decryption, especially DNN black boxes and perceptual hash black boxes, which are mostly adapted for the purpose of classification or similarity detection.

(3) Although it is difficult to obtain the target input of IUM, by establishing the activity chain from seeding inputs to obtaining behavioral results, advanced reverse analysis experts can conclude that the cipher binary will release an executable file once successfully decrypted, and whether the cipher binary is decrypted or not seems to affect the function of its parasitic benign program, which is a necessary condition to assist in determining the existence of an attack payload.

Separate detection for any of the above features is not enough to identify malware. However, because the above features can be captured by means of behavior monitoring or reverse analysis, this paper suggests that graph analysis-based method (as shown in Figure 7) can be leveraged to aggregate these features and build a distinct behavior sequence, so as to improve the confidence of malware detection results, and reduce the cost of manual screening non-malware. Besides, cyber threat intelligence (CTI) [42] injects new vitality into malware detection. It is also advised to update the behavior sequence of IUM into open-source CTI databases, which is promising to realize IUM detection in any CTI-integrated environment.

6.2 The behavioral monitoring-based heterogeneous defense method

As an environment keying malware, early information collection is necessary for the construction of IUM. For example, robust DNN models rely on the collection of face image information; accurate and effective attack payload depends on the collection of information about the target defense environment.

In this regard, this paper suggests that a heterogeneous defense model based on behavior monitoring can be constructed on the security defense side. As shown in Figure 8, a behavior monitor is set at the target environment. Once there is a collection behavior of critical environment information (①), the behavior monitor will perceive (②) and start a heterogeneous update (③) of the target defense environment to invalidate the information acquired by the attacker. Thus, when an attacker performs malware delivery (④), the probability that the attack payload fails to escape the new defenses will greatly increase.

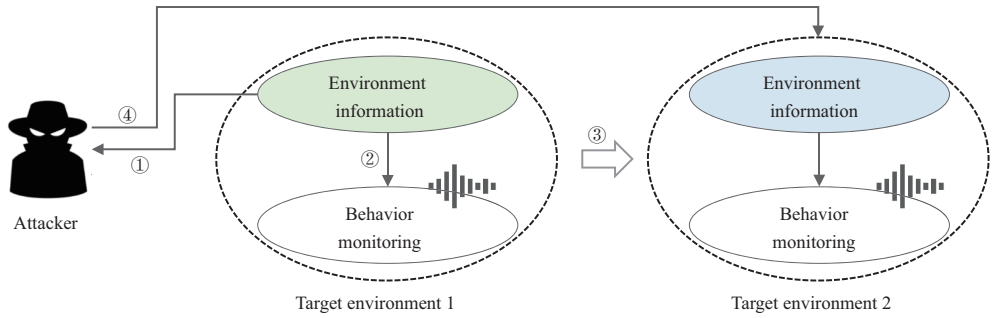


Figure 8 (Color online) Heterogeneous defense architecture based on behavior monitoring.

Table 11 Overview of evasive malware

Research type	Action stage	Ref.	Feature
Against neural networks in the problem space	Delivery	[43, 44]	Establish the mapping of the feature space to the problem space
Condition code confusion	Persistence	[45]	Input customization, conditional code confusion
Thread injection	Runtime	[46]	Use the APC to inject malware chunks into infected existing threads
Advanced packing	Delivery&Persistence	[47]	Release the encrypted Dex Data in advance, or with the runtime self-modification
IUM	Delivery&Persistence	–	Attack targets are customized, and the integrated unknown attack techniques can be reused

7 Related work

This paper surveys some of the current advanced defense evasion techniques and summarizes them in Table 11. In terms of the action stage, we focus on comparing IUM with the malware based on condition code confusion and the malware based on an advanced package.

7.1 Condition code confusion

In 2008, Sharif et al. [45] proposed a conditional code obfuscation method for trigger-based malware to hide malicious behavior. In this method, a compiler-level tool is developed. Through the steps of source code parsing, conditional code conversion, code generation and code decryption, an obfuscated binary file can be automatically generated for the open-sourced malware, and the key used to decrypt the obfuscated binary file does not exist in the malware. Instead, it relies on the correct input provided at runtime which can be taken as the key to decrypt execution since the correct input satisfies the code conditions. Ideally, this kind of malware would work against all static analysis tools as well as an input-oblivious dynamic analyzer. However, there are limited types of conditional codes that can be confused by this method, and the encryption strength depends on the variables used in conditional codes, so brute force cracking and dictionary attacks are possible.

Different from their methods, IUM uses the idea of homemade trigger to encrypt payload/original malware. IUM hosts these payloads with benign applications and does not care about how they are implemented at the code level, i.e., without qualifying the type of packaged payload/original malware, thus shedding the limitations of the conditional code confusion method described above. In addition, the target of the conditional code confusion method is only to hide malicious behaviors. In contrast, IUM binds "intention hiding" with the "accurate identification" function, which helps IUM realize the hiding of attack targets (classes and instances), expanding the hiding range of malicious intentions.

7.2 Advanced package

In order to better realize the ability to evade detection in the delivery or persistence stage, some malware tries to hide malicious intention by using the idea of software reinforcement (especially Android APP protection), which makes it difficult to analyze and determine, and expands the scope of malware dissemination to a certain extent. "DEX encryption protection" and "runtime dynamic self-modification" are advanced packing technologies that can be abused by attackers, and malware uses these techniques

to make it effective against static analysis; however, since such malware did not tightly couple the function of "intention concealment" with the target of "precision attack", many researchers have proposed dynamic analysis methods for this kind of malware. For example, in 2021, Happer, a tool to unpack such malware through hardware assistance, was proposed by Xue et al. [47]. By monitoring the runtime behavior of malware and exploiting the hardware features of the ARM platform, Happer can realize the internal dump of the Dex data hidden by malware, thus allowing the malware to release all the hidden Dex data at runtime, and finally realize the effective recovery of the original Dex files.

By contrast, IUM achieves a tight coupling between the function of "intention concealment" and the target of "precision attack", which prevents dynamic analysis tools from obtaining the key to release critical data (malicious payload). Therefore, the analyzer cannot recover the key data of IUM, thus ensuring that IUM can always hide the attack intention from detection in the non-target scene. Through the accurate identification of the target, it can release the key data when and only when the target is found, so as to implement the accurate attack. This feature of IUM allows it to be mainly used in APT attacks, which also coincides with the fact that our attack cases in the wild are all derived from APT attacks.

8 Conclusion

To be well prepared for future malware attacks, we should study advanced forms of malware that could soon be developed by attackers. In this paper, we present a new type of malware threat, IUM. Compared to current malware, IUM is much harder to be detected, and even if it is detected as malicious by advanced detection tools, it will not expose the hidden attack intention, so that a class of malware with the same attack intention cannot be completely detected. This paper provides a framework for a systematic understanding of IUM, which analyzes IUM from the aspects of formal definition, construction process, security property evaluation, and key technology realization.

In particular, for the design of IUM's core components iKeyGen and iKeyDis, this paper provides two black box characteristics and summarizes the current existing related research into four black box design schemes. Among these schemes, the perceptual hash black box is a breakthrough work first proposed by this paper. In addition, by considering the volume increase, these four types of black boxes can help complete the batch generation and practical application of IUM, which further emphasizes the seriousness of the new security threat represented by IUM.

Finally, this paper provides insights into defense against IUM, which is designed to serve the construction and improvement of defense systems. In future research work, we will explore more defense technologies against IUM. Meanwhile, although the PoCs of IUM have successfully countered the mainstream detection and security assessment mechanisms such as Kaspersky, Windows Defender, VirusTotal, and ThreatBook, we are still concerned about the power of IUM in practice. Therefore, the actual attack capability of IUM must be measured in offensive and defensive cyber ranges.

Acknowledgements This work was supported by Key-Area Research and Development Program of Guangdong Province (Grant Nos. 2019B010136003, 2019B010137004) and National Key Research and Development Plan of China (Grant No. 2019YFA0706404).

Supporting information Appendixes A–C. The supporting information is available online at info.scichina.com and link.springer.com. The supporting materials are published as submitted, without typesetting or editing. The responsibility for scientific accuracy and content remains entirely with the authors.

References

1 Botacin M, da Rocha V F, de Geus P L, et al. Analysis, anti-analysis, anti-anti-analysis: an overview of the evasive malware scenario. Anais do XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais, 2017, 17: 250–263
2 Wang Q, Hassan W U, Li D, et al. You are what you do: hunting stealthy malware via data provenance analysis. In: Proceedings of the 27th Annual Network and Distributed System Security Symposium (NDSS). San Diego: The Internet Society, 2020. 1–17
3 Wagner D, Soto P. Mimicry attacks on host-based intrusion detection systems. In: Proceedings of the 9th ACM Conference on Computer and Communications Security. New York: Association for Computing Machinery, 2002. 255–264
4 Fogla P, Sharif M I, Perdisci R, et al. Polymorphic blending attacks. In: Proceedings of the 15th USENIX Security Symposium. Berkeley: USENIX Association, 2006. 241–256
5 Bonfante G, Fernandez J, Marion J-Y, et al. 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. New York: Association for Computing Machinery, 2015. 745–756
6 Li Z T, Li W L, Lin F Y, et al. Hybrid malware detection approach with feedback-directed machine learning. Sci China Inf Sci, 2020, 63: 139103
7 Wingfield T. Fileless Malware Execution With Powershell is Easier Than You May Realize. McAfee Technical Report, 2017. [2021-12-16]. https://www.mcafee.com/enterprise/en-us/assets/solution-briefs/sb-fileless-malware-execution.pdf

8 GOODIN D. A rash of invisible, fileless malware is infecting banks around the globe. ARS Technica, 2017. [2021-12-16]. https://arstechnica.com/information-technology/2017/02/a-rash-of-invisible-fileless-malware-is-infecting-banks-around-the-globe/?comments=1&post=32786675
9 Larry. The 2017 State of Endpoint Security Risk Report. Ponemon Institute Technical Report, 2017. [2021-12-16]. https://cdn2.hubspot.net/hubfs/468115/Campaigns/2017-Ponemon-Report/2017-ponemon-report-key-findings.pdf
10 MITRE. Process hollowing. 2020. [2021-12-16]. https://attack.mitre.org/techniques/T1093/
11 GREAT. The mystery of the encrypted Gauss payload. 2012. [2021-12-16]. https://securelist.com/the-mysteryof-the-encrypted-gauss-payload-5/33561/
12 Ishimaru S. Why corrupted (?) samples in recent APT? 2016. [2021-12-16]. https://hitcon.org/2016/pacific/0composition/pdf/1201/1201
13 Kirat D, Jang J, Stoecklin M P. DeepLocker — concealing targeted attacks with AI locksmithing. In: Proceedings of the Black Hat Conference, Las Vegas, 2018. 1–29
14 MITRE. MITRE ATT&CK — Software. 2015. [2021-12-16]. https://attack.mitre.org/software/
15 Strom B E, Applebaum A, Miller D P, et al. MITRE ATT&CK: Design and Philosophy. Technical Report. 2018
16 Martin Lockheed. The cyber kill chain. 2011. [2021-12-16]. https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html
17 Yadav T, Rao A M. Technical aspects of cyber kill chain. In: Proceedings of the 3rd International Symposium on Security in Computing and Communication. Cham: Springer, 2015. 438–452
18 NSA. NSA/CSS technical cyber threat framework V2. National Security Agency Cybersecurity Report, 2018. [2021-12-16]. https://media.defense.gov/2019/Jul/16/2002158108/-1/-1/0/CTR_NSA-CSS-TECHNICAL-CYBER-THREAT-FRAMEWORK_V2.PDF
19 Moran N, Bennett J T. Supply Chain Analysis: From Quartermaster to Sunshop. FireEye Technical Report, 2013
20 Peisert S, Schneier B, Okhravi H, et al. Perspectives on the SolarWinds Incident. IEEE Secur Privacy, 2021, 19: 7–13
21 Oxford Analytica. Kaseya ransomware attack underlines supply chain risks. Emerald Expert Briefings, 2021. doi: 10.1108/oxan-es262642
22 Engelberg J. Bash Uploader Security Update. Technical Report, 2021. [2021-12-16]. https://about.codecov.io/security-update/
23 Antonakakis M, Perdisci R, Dagon D, et al. Building a dynamic reputation system for DNS. In: Proceedings of the 19th USENIX Security Symposium. Berkeley: USENIX Association, 2010. 273–290
24 Wang K, Stolfo S J. Anomalous payload-based network intrusion detection. In: Proceedings of the International Workshop on Recent Advances in Intrusion Detection. Berlin: Springer, 2004. 203–222
25 Zauner C. Implementation and benchmarking of perceptual image hash functions. Computer Science, 2010. [2021-12-16]. https://www.phash.org/docs/pubs/thesis_zauner.pdf
26 Özyurt F, Tuncer T, Avci E, et al. A novel liver image classification method using perceptual hash-based convolutional neural network. Arab J Sci Eng, 2019, 44: 3173–3182
27 Google. Google images. [2021-12-16]. https://www.google.com/imghp?hl=en
28 Liu Z, Luo P, Wang X, et al. Deep learning face attributes in the wild. In: Proceedings of the IEEE International Conference on Computer Vision. Washington: IEEE Computer Society, 2015. 3730–3738
29 Ng H-W, Winkler S. A data-driven approach to cleaning large face datasets. In: Proceedings of 2014 IEEE International Conference on Image Processing (ICIP). Washington: IEEE Computer Society, 2014. 343–347
30 Parkhi O M, Vedaldi A, Zisserman A. VGG face descriptor. Dataset, 2015. [2021-11-16]. https://www.robots.ox.ac.uk/~vgg/software/vgg_face/
31 Fanelli G, Dantone M, Gall J, et al. Random forests for real time 3D face analysis. Int J Comput Vis, 2013, 101: 437–458
32 Wolf L, Hassner T, Maoz I. Face recognition in unconstrained videos with matched background similarity. In: Proceedings of the IEEE Computer Vision and Pattern Recognition (CVPR). Washington: IEEE Computer Society, 2011. 529–534
33 Kumar N, Berg A C, Belhumeur P N, et al. Attribute and simile classifiers for face verification. In: Proceedings of IEEE 12th International Conference on Computer Vision. Washington: IEEE Computer Society, 2009. 365–372
34 Huang G B, Mattar M, Berg T, et al. Labeled faces in the wild: a database for studying face recognition in unconstrained environments. In: Proceedings of International Workshop on Faces in Real-Life Images: Detection, Alignment, and Recognition, 2008. 1–17
35 SophosLabs Research Team. Emotet exposed: looking inside highly destructive malware. Network Secur, 2019, 2019: 6–11
36 Ramos E. Analysis: Ursnif-Spying on Your Data Since 2007. Technical Report, 2016. [2021-12-16]. https://www.gdatasoftware.com/blog/2016/11/29325-analysis-ursnif-spying-on-your-data-since-2007
37 Holland A. Spot the difference: tracking malware campaigns using visually similar images. 2019. [2021-12-16]. https://threatresearch.ext.hp.com/spot-the-difference-trackingmalware-campaigns-using-visually-similar-images/
38 Google. VirusTotal. 2007. [2021-12-16]. https://www.virustotal.com/
39 ThreatBook. Threatbook cloud sandbox. 2015. [2021-12-16]. https://s.threatbook.cn/
40 Shalev S. theZoo — a live malware repository. 2014. [2021-12-16]. https://github.com/ytisf/theZoo
41 OPSWAT. Metadefender cloud. 2002. [2021-12-16]. https://metadefender.opswat.com/
42 He Y, Inglut E, Luo C J. Malware incident response (IR) informed by cyber threat intelligence (CTI). Sci China Inf Sci, 2022, 65: 179105
43 Pierazzi F, Pendlebury F, Cortellazzi J, et al. Intriguing properties of adversarial ML attacks in the problem space. In: Proceedings of 2020 IEEE Symposium on Security and Privacy (SP). Los Alamitos: IEEE COMPUTER SOC, 2020. 1332–1349
44 Zhao K, Zhou H, Zhu Y-L, et al. Structural attack against graph based android malware detection. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. New York: Association for Computing Machinery, 2021. 3218–3235
45 Sharif M I, Lanzi A, Giffin J T, et al. Impeding malware analysis using conditional code obfuscation. In: Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS). San Diego: The Internet Society, 2008. 1–13
46 Pavithran J, Patnaik M, Rebeiro C. D-TIME: distributed threadless independent malware execution for runtime obfuscation. In: Proceedings of the 13th USENIX Workshop on Offensive Technologies (WOOT 19). Berkeley: USENIX Association, 2019. 1–14
47 Xue L, Zhou H, Luo X-P, et al. Happer: unpacking Android apps via a hardware-assisted approach. In: Proceedings of 2021 IEEE Symposium on Security and Privacy (SP). Los Alamitos: IEEE Computer Soc, 2021. 1641–1658