Environmental Authentication in Malware

Jeremy Blackthorne[1], Benjamin Kaiser[2], and Benjamin Fuller[3], and Bülent Yener[1]

[1] Rensselaer Polytechnic Institute, Troy NY 12151, USA
{whitej12,byener}@rpi.edu,
[2] benjamin.h.kaiser@gmail.com
[3] University of Connecticut, Storrs, CT 06269, USA
benjamin.fuller@uconn.edu

Abstract. Malware needs to execute on a target machine while simultaneously keeping its payload confidential from a malware analyst. Standard encryption can be used to ensure the confidentiality, but it does not address the problem of hiding the key. Any analyst can find the decryption key if it is stored in the malware or derived in plain view.
One approach is to derive the key from a part of the environment which changes when the analyst is present. Such malware derives a key from the environment and encrypts its true functionality under this key.
In this paper, we present a formal framework for *environmental authentication*. We formalize the interaction between malware and analyst in three settings: 1) blind: in which the analyst does not have access to the target environment, 2) basic: where the analyst can load a single analysis toolkit on an effected target, and 3) resettable: where the analyst can create multiple copies of an infected environment. We show necessary and sufficient conditions for malware security in the blind and basic games and show that even under mild conditions, the analyst can always win in the resettable scenario.

Keywords: environmental keying, environmental authentication, malware

1 Introduction

In many settings, programs try to prevent observers from learning their behavior. These settings vary from legitimate software protecting its intellectual property through digital rights management to malware hiding from analysts to extend the life of a criminal endeavor.

We focus on malware hiding from an analyst, but our discussion applies to the other scenarios as well. Our goal is to improve the understanding of current and future malware techniques. Our work proceeds from the point of view of the malware hiding from an adversarial analyst. Thus, our discussion reverses roles: the malware designer is the party trying to ensure security and the analyst acts as the adversary.

Malware follows two approaches to hiding its behavior: 1) making the observed program unintelligible, i.e. *obfuscation* [CTL97, BGI+01, GGH+13], and 2) preventing observation from even occurring when executing in the wrong environment, i.e. *environmental authentication* [RS98, SRL12].

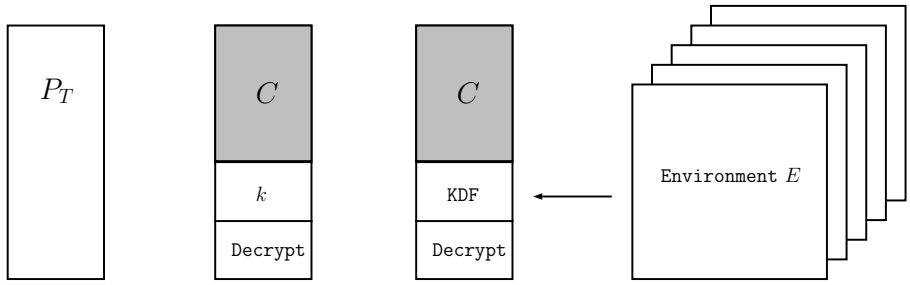
Obfuscation is the subject of informal [CTL97] and formal [BGI+01] treatments. Obfuscation works as follows: an obfuscator function $\mathcal{O}(.)$ takes some program P as input and creates P' such that P' is input-output equivalent to P but is implemented differently. The implementation is changed with the goal of confusing an analyst which tries to understand the program. But even the strongest obfuscation scheme cannot hide important aspects of the program including input/output behavior. Some functions can be recovered by just observing a polynomial number of input-output pairs [SWP08]. Such functions are known as learnable. For malware, the desire is to hide the effects on the target computer system, the inner workings of the algorithm are a secondary concern. For this stronger level of protection, malware attempts to prevent observation from occurring. Malware achieves this by distinguishing environments in which it is being observed from environments which it is not. This distinguishing of environments we call *environmental authentication*.

Environmentally authenticating malware targets a particular computer (or set of computers) and learns as much as possible about this *target* environment. It then creates (at least) two distinct behaviors: one for the target environment and another for non-target or observed environments. At runtime, the malware determines its current executing environment and executes the appropriate behavior [BCK+10a]. Environmental authentication can be subdivided into two approaches: 1) environmental sensitivity and 2) environmental keying.

Environmental Sensitivity Environmentally sensitive malware reads system state and incorporates this state into program control flow [BKY16]. As an example, the Windows API includes a function `IsDebuggerPresent` which allows a program to detect if a user level debugger is instrumenting their program. Many pieces of malware change their behavior based on the value of this call. This approach makes a binary and observable decision on how the environment affects control flow. This means that an analyst can run a debugger, create a breakpoint at this system call, and manually overwrite the return to be true. This corresponds to a weak form of authentication (also known as binary matching [ICF+15]).

This has lead to an arms race between malware trying to sense the presence of analysis techniques and analysis techniques trying to create small and unobservable changes in the system state. Malware authors created techniques to detect debuggers [CAM+08, Fer11, SH12], virtual machines [Fer07, SH12], and system emulators [KYH+09, PMRB09]. All environmental sensing techniques make binary decisions based on the environment.

Environmental Keying Environmental keying replaces the binary decision of environmental sensing with key derivation. This approach is performed in three stages:



(a) Target Payload (b) Encrypted (c) Encrypted with environmental keying

Fig. 1: A plaintext payload P_T is shown in (a) as a baseline. In (b) we see the same payload P_T transformed into an encrypted version C. The encrypted payload must include an unencrypted key and a decryption function. In (c) we see the same encrypted payload from (b) with k replaced with the `KDF` function. `KDF` takes the environment E as an input and derives k as output. In this figure the alternate payload P_O is removed for clarity.

1. The malware author targets a computer (or class of computers). Information about the target computer is observed and recorded in the malware. In addition, the author gathers information about other configurations which can be considered as invalid or under *observation*.
2. The author derives cryptographic keys from the target environment and observed environments.[4]
3. The author encrypts different program behaviors under each of these keys and adds a key derivation process to switch between these behaviors.

At run time, the malware measures the environment and derives a key from this environment. Environmentally keyed malware is split into three functionalities: a key derivation function (KDF) and encrypted payloads P_T and P_O corresponding to the desired behavior in the target and observed environments respectively. When deployed, the malware first derives a key from the environment and then try to decrypt each payload. This process of unlocking functionality is shown in Figure 1. For example, the malware Gauss derives a key from its environment by computing an MD5 hash 10,000 times over a combination of the %PATH% variable and the directory names in %PROGRAMFILES% [RT12]. To the best of our knowledge, Gauss' target behavior has not been decrypted.

[4] Extractors [NZ96] and fuzzy extractors [DRS04] can be used to derive keys in non-noisy and noisy environments, respectively. See the works of Nisan and Ta-Sha [NTS99] and Dodis et al. [DRS08] respectively for more information. Throughout this work we assume that the key derivation techniques are implemented properly and the only weakness that can be targeted is guessing a valid input to the key derivation process.

Encryption prevents an analyst from reasoning about the target payload P_T. There is no binary decision that can be flipped by an analyst to force the malware to decrypt the payload in an incorrect environment. There are two main questions in this setting, 1) can the malware designer find high entropy sources for key derivation, 2) can the analyst observe the malware without disturbing these sources.

Obfuscation has a long history in both the systems and theoretical computer science communities. Environmental authentication, on the other hand, is known in the systems community but unexplored from a theoretical perspective. The malware community is rapidly adopting new techniques, forcing analysts to scramble to develop new analysis capabilities in order to keep up. The development of a theoretical foundation for environmental authentication will empower analysts to develop more effective tools for analyzing malware that uses environment authentication.

Our Contribution We put forth a formal model for environmental authentication and evaluate three common malware analysis settings:

Section 3 An analyst that does not have access to the target environment to which the malware is keyed. We call this the *blind* setting.

Section 4 The analyst has access to the environment after the malware has infected it and cannot create an offline backup of the system. This setting represents an analyst performing incident response on a critical system. For example, a controller at a power plant cannot be taken offline. We call this the *basic* setting.

Section 5 The analyst is able to snapshot an infected system. They are able to create multiple copies and install different analysis tools on each copy. We call this the *resettable* setting.

In all settings a piece of malware M interacts with the environment E through a series of decision algorithms, $\mathcal{D}_1, ..., \mathcal{D}_n$, which read subsets of the environment to determine the execution path. Recalling the stages of environmental keying: the decision algorithms represent measuring the environment, deriving a key, and attempting to decrypt the next section of the program. We do not allow the analyst observe the code of the current decision algorithm or beyond so 1) our results hold in the presence of obfuscation and 2) because any code beyond the decision procedure may be encrypted. The analyst's primary means of interacting with the malware is by providing inputs to the decision algorithms, which represents altering the environment (as the input to each decision algorithm is a reading of the environment). The (informal) goal of M is to satisfy correctness and soundness:

Correctness M achieves *c*orrectness if it reaches the payload stage P_T in the target environment.

Soundness M achieves *s*oundness if it never reaches the payload stage P_T when the analyst A is present in the environment.

	Necessary	Sufficient
Blind	Thm 2: Some $\mathcal{D}_i$ outputs 1 with `negl` probability on random inputs.	Thm 3: Some $\mathcal{D}_i$ outputs 1 with `negl` probability on best case inputs
Basic	Thm 4: Decision procedure and analyst likely to overlap	Thm 5: Most of environment is entropic

Table 1: Summary of results. Necessary and sufficient conditions are from the point of view of the malware designer. The resettable setting is omitted as M security is not possible in this setting.

We provide necessary and sufficient conditions for M to be secure in the blind and basic games. In the resettable game, we show that under very mild assumptions, the analyst always wins.

Our results for the blind game are intuitive: for M to be secure, it is necessary that a decision procedure rarely outputs 1 in a random environment. It is sufficient that there does not exist a "worst case" environment that can cause a random decision procedure to regularly output 1. This means that in practice, decision procedures must be precisely keyed to their target environment.

Our results for the basic game are more complicated. In this setting, the analyst may read the target environment but first has to load an analysis technique. This process of loading can overwrite some critical part of the environment. A necessary condition for security is for the analysis technique to be likely to overwrite a large subset of the environment that will be used in some decision procedure. A sufficient condition for security is that this subset is likely to be "entropic", i.e., there are few values for it that cause a decision procedure to accept. The first condition is intuitive, but the second conflicts somewhat with our understanding of computers, for although we don't know the distribution of all aspects of a computer system, it seems unlikely to be large for all subsets.

For the resettable game, we provide a simple proof that the analyst can learn the entire target environment, and thus environmental keying provides little security. Our results for the blind and basic settings are summarized in Table 1. We note our results are information-theoretic as we assume that the A only has oracle access to decision procedures.

1.1 Other Related Work

Protecting programs by depending on the environment has been studied under many names, including environmental key generation [RS98], secure triggers [FKSW06], host-based fingerprinting [KLZS12], environment-sensitive malware [LKMC11, SRL12], host-identity based encryption [SRL12], environment-targeted malware [XZGL14], malware with split personalities [BCK+10b], and environmental keying [Moo15, Bau14]. We use the term *environmental authentication* to describe any technique that creates a dependence on a specific environment or type of environment for the purposes of preventing observation or analysis.

Transparent analysis analyzes programs while minimizing detectable environmental changes [Yan13]. Dinaburg et. al present a formalization for transparent malware analysis in [DRSL08] and describe the requirements for transparent analysis. Their requirements are higher privilege, the absence of side-channels, transparent exception handling, and identical timings. Kang et. al also formulate the problem of transparent malware analysis within emulators [KYH+09].

Key derivation is a sub-field of cryptography that studies ways to extract uniformly random strings from high-entropy, non-uniform sources [Kra10]. Deriving keys in the presence of noise is often necessary for real-world applications and is achieved by fuzzy extractors [DRS04]. Throughout this work we assume that key derivation is ideal, a resulting key is secure if it results from super-logarithmic min-entropy. In the noise-free setting, this is sufficient in the random oracle model [BR93]. This may not be sufficient in the noisy case, a more precise notion is fuzzy min-entropy [FRS16], we ignore these losses in this work.

Organization The rest of the paper is organized as follows: in Section 2, we provide the necessary background information, notation, and preliminary definitions, including the formal definition of environmental authentication. In Sections 3, 4 and 5, we describe the blind, basic, and resettable settings respectively.

2 Definitions

Functions are written in the typewriter font, e.g. `Function`, distributions using script font and a single letter, e.g. $\mathcal{D}$, and scalar values using math font with a single lowercase letter, e.g. k. If k is sampled from a distribution $\mathcal{D}$, we say $k \leftarrow \mathcal{D}$. If k is an element in a set K, we say $k \in K$.

2.1 Modeling Computer Systems.

Computer systems are complex, as programs can read state from a variety of sources: memory, hard drive, cache, side-channels, operating system calls, registers, installed devices, network interfaces, and more. Turing machines and interactive Turing machines do not capture all of this interaction, particularly for two programs operating in the same system.

The goals of malware are 1) *correctness*: detecting if they are resident on a target set of machines and 2) *soundness*: discerning if the system is being analyzed. These goals can be modeled by abstracting various device state into a single array E which we call the environment. The two goals can be stated as:

Correctness The malware should read enough of E to be sure it is on a targeted machine. In particular, it should read features that vary between devices. During targeting it is necessary for the designer to learn the relevant features of the target set.

Soundness The malware should read parts of the array that are likely to change under observation. As mentioned in the introduction, parts of the array that change under observation include `IsDebuggerPresent` (which is easy to hide) and timing side-channels (which are harder to hide).

The goal of the analyst is to understand both E and the malware M without causing changes to E. In pursuing this goal, we assume that the analyst has two main capabilities

1. They are able to create (representative) computer systems and read all of E.
2. If the analyst has access to the target computer they can read from the environment after being loaded on the system. This action may cause detectable and irreversible changes to E.

We now formalize the correctness goal of malware. We defer soundness to the following sections as we consider it with regards to multiple analysis postures.

Model A computer system is a one-dimensional array E of length ℓ ($E \in \{0,1\}^\ell$). We denote by $\mathcal{E}$ the distribution of possible system environments and a single computer system E is sampled from $\mathcal{E}$ ($E \leftarrow \mathcal{E}$). Either the malware author or analyst may have more information about the target environment or the overall distribution of computer systems. For instance, the malware designer may be targeting an English language system while this is unknown to the analysts. Our model should extend to this setting but we leave this formalization as future work.

All algorithms are executed in the environment but must be loaded into E via the `Load` function. This (irreversibly) changes the environment E into E'. Only after being loaded can an algorithm read from or write to the environment. When M is loaded onto E, denoted $\texttt{Load}(M, E)$, its goal is to authenticate the environment using a sequence of decision algorithms $\mathcal{D}_i$ and sensors S_i. A sensor S_i is a subset of $[1..\ell]$. The corresponding decision algorithm $\mathcal{D}_i(E'_{S_i})$ takes as input the environment at the set of locations $\{E_j | j \in S_i\}$. $\mathcal{D}_i$ outputs 1 to indicate the environment matches the target environment (i.e. continue on a execution path that allows it to deliver its target payload) and 0 otherwise.

We assume this payload is of minimal size in comparison to the environment and thus we do not include it in the model. The analyst wins if they pass all decision procedures. Authentication decisions may be implicit through the use of cryptographic authentication, thus we only allow an analyst to provide inputs to $\mathcal{D}_i$ in a black-box manner and decision algorithms output a binary decision. There is no way to force the decision procedure to output a 1.

Limitations of our model Computer systems change over time. We do not model time for an analyst because a determined analyst can control the system environment and essentially stop time. In real computer systems, the malware can only read a single address at a time which is either 32 or 64 bits. Several of our results will depend on the size of memory that M can read in a single decision algorithm, we call this parameter readsize or α. We assume that α is substantially larger than a single memory location. It is an interesting open problem to extend our results to a setting where a decision procedure cannot read all of its input in a single timestep.

Correctness For malware to authenticate its environment it must be *correct*, meaning that it executes its payload in its intended environment, and *sound*, meaning that it does not reveal its payload in an observed environment. We present a definition for correctness here and define three soundness definitions in the following sections. First, however, we must describe precisely how sensitive a piece of malware with both correctness and soundness is against an analysis technique A. We capture this property in the following definition.

We define correctness with the following game:

Experiment $\mathbf{Exp}^{\text{cor}}_{M,E}$:
$(\mathcal{D}_1, S_1, ..., \mathcal{D}_n, S_n) \leftarrow M(\cdot)$
$E' \leftarrow \texttt{Load}(M, E)$
If $\forall i$, $[\mathcal{D}_i(E'_{S_i}) = 1]$ return 1
Else return 0.

Denote by the parameter n the number of decision algorithms and α the maximum size of S_i. We assume that each $\mathcal{D}_i$ is deterministic and the probability is over the coins of M and $\texttt{Load}$.

Definition 1. *A piece of malware M is δ-correct on E if* $\Pr[\mathbf{Exp}^{\text{per}}_{M,E}(\cdot) = 1] = \delta$.

Environment Samplability We assume the analyst is able to read the state of representative computers and may be able to load on the targeted computer with the malware present. We now formalize this first capability:

Assumption 1 *There exists a randomized algorithm $\texttt{Sam}_E$ running in time $t_{\mathcal{E}}$ such that $\texttt{Sam}_E(\cdot) \stackrel{d}{=} \mathcal{E}$.*

If the malware accepts frequently on random computers there is no need for the analyst to understand the target environment. That is, access to the target environment is not necessary if the decision procedures output 1 frequently on random computers:

Definition 2. *Define the* accepting probability *of M over n possible environments, denoted*
$\texttt{Accept}(M, \mathcal{E})$*, as*

$$\texttt{Accept}(M, \mathcal{E}) = \min_{1 \le i \le n} \left(\mathbb{E}_{E \leftarrow \mathcal{E}} \left(\Pr_{\mathcal{D}_i, S_i \leftarrow M} [\mathcal{D}_i(E_{S_i}) = 1] \right) \right).$$

Accepting probability captures how frequently the malware succeeds on a random computer system. However, it may be possible for an analyst to learn more information by observing the behavior of the previous decisions procedures. To capture this notion we present the following (information-theoretic) definition:

Definition 3. *Define the* adaptive guessing probability *of M over n possible environments, denoted as* $\texttt{AGuess}(M, \mathcal{E})$*, as*

$$\texttt{AGuess}(M, \mathcal{E}) = \min_{1 \leq i \leq n} \left(\max_{E' \in \mathcal{E}} \left(\Pr_{\mathcal{D}_i, S_i \leftarrow M} [\mathcal{D}_i(E'_{S_i}) = 1 | \mathcal{D}_1, ..., \mathcal{D}_{i-1}] \right) \right).$$

where $\mathcal{D}_i$ is the entire truth table of $\mathcal{D}_i$.

These definitions capture security against an analyst trying random computer systems and an analyst finding the best computer system respectively. They can be thought of as analogues of Shannon and min-entropy respectively [Rén61]. We do not condition on the previous decision algorithms in Definition 2 as this does not change the expectation but this could be included without affecting `Accept`.

Definition 4. *M is* (β, γ)-environmentally authenticating *if:*

- $\texttt{Accept}(M, \mathcal{E}) \geq 2^{-\beta}$.
- $\texttt{AGuess}(M, \mathcal{E}) \leq 2^{-\gamma}$.

Proposition 1. $\texttt{AGuess}(M, i, \mathcal{E}) \geq \texttt{Accept}(M, i, \mathcal{E})$ *and thus for any* (β, γ)*-environmentally authenticating malware* $\gamma \leq \beta$.

With these definitions we can formalize the notion of environmental sensitivity and environmental keying described in the introduction.

Definition 5. *Let λ be a security parameter. If M is (β, γ) environmentally authenticating for $\beta = O(\log \lambda)$ then M is* environmentally sensing.

Definition 6. *Let λ be a security parameter. If M is (β, γ) environmentally authenticating for $\gamma = \omega(\log \lambda)$ then M is* environmentally keying.

By Proposition 1 $\gamma \leq \beta$, thus malware cannot be both environmentally sensing and environmentally keying. There is malware that is neither environmentally sensing nor environmentally keying.

3 Blind Scenario

The first adversarial scenario models malware being found in the wild separate from its target environment. This is common in real malware, which may spread widely and infect many machines beyond its target, if it even has a specific target. This separation of malware and target environment is important when attempting to understand malware with environmental authentication. In this scenario, the analyst does not know or have access to the target environment, we also assumes that the analyst cannot determine the target environment by reverse engineering the malware; this scenario is demonstrated in practice by the malware Gauss, for which a target environment has not been found despite significant effort by the analysis community [RT12].

We define blind soundness using the following game:

Experiment $\mathbf{Exp}^{\text{bli-sou}}_{M,E,A}$:
$(\mathcal{D}_1, S_1, ..., \mathcal{D}_n, S_n) \leftarrow M$
For $i = 1$ to n
  $\texttt{Guess}_i = A^{\mathcal{D}_i(\cdot)}(S_i, \mathcal{D}_{i-1}, S_{i-1}, ..., \mathcal{D}_1, S_1)$.
If $\forall i$, $\mathcal{D}_i(\texttt{Guess}_i) = 1$ return 1
Else return 0.

In this game, A receives a complete description of all prior decision algorithms and the current sensor readings. They also have oracle access to the current decision procedure. We denote by t_{oracle} the time needed to make an oracle call and assume this time is consistent across decision procedures.

Definition 7. *M is ϵ-blind sound against A if* $\Pr[\mathbf{Exp}^{\text{bli-sou}}_{M,E,A}(\cdot) = 1] < \epsilon$.

Our results in the blind game are intuitive. A necessary condition for soundness is that `Accept` accepts with negligible probability on random inputs. A sufficient condition for soundness is that `AGuess` accepts with negligible probability on worst case inputs.

Theorem 2. *For any (β, γ)-environmentally authenticating malware $\mathcal{M}$ with n decision procedures that is at most $1 - \delta$ correct, for any $0 < \epsilon < 1$ there exists A such that M is at most $(\epsilon + \delta)$-blind sound where A runs in time*

$$t_A = 2^{\beta} n (t_E + t_{oracle}) \ln\left(\frac{n}{1-\epsilon}\right)$$

The proof of this theorem can be found Appendix A.1. At a high level, the A can sample environments randomly until each decision procedure accepts. The result implies that environmentally sensitive malware is not sound in the blind game:

Corollary 1. *Let λ be a security parameter, if M is environmentally sensing (i.e. $2^{\beta} = \texttt{poly}(\lambda)$) and $1 - \delta$ correct, and $n, t_{oracle}, t_E = \texttt{poly}(\lambda)$ for any $\epsilon \leq 1 - 2^{-\texttt{poly}(\lambda)}$, there exists an A that runs in time $\texttt{poly}(\lambda)$ such that M is at most $\epsilon + \delta$ sound.*

We further show having a high γ suffices for security in the blind game.

Theorem 3. *For any (β, γ, n)-environmentally authenticating malware $\mathcal{M}$ that is $1 - \delta$-correct, let A be a block-box algorithm that makes at most t calls to the decision oracles, then M is at least ϵ-sound for $\epsilon = (t+1)\frac{2^{-\gamma}}{1-t2^{-\gamma}}$.*

The proof of this theorem can be found in Appendix A.2. At a high level, since a decision procedure has a negligible probability of accepting, even with a polynomial number of guesses the overall acceptance probability remains negligible in the security parameter. The result implies that all environmentally keyed systems are secure in the Blind game:

Corollary 2. *Let λ be a security parameter, if M is environmentally keying, then for any black-box A making $t = \texttt{poly}(\lambda)$ oracle calls, $\epsilon = \texttt{negl}(\lambda)$.*

Proof. The proof proceeds by noting that for $t = \mathtt{poly}(\lambda)$ and $2^{-\gamma} = \mathtt{negl}(\lambda)$ then $1 - t2^{-\gamma} \geq 1/2$ and thus $\epsilon \leq 2(t+1)2^{-\gamma} = \mathtt{negl}(\lambda)$.

Without access to the intended environment E, the blind adversary is at a significant disadvantage. As long as the key has sufficient entropy, the scheme is sound. We see a real example of this in the malware Gauss. Almost four years after Gauss was first reported [RT12], we see that there still have been no public success in deciphering its payload. There has even been developed an open source, distributed cracker developed to harness global computing power to solve the mystery without success [Jst16].

4 Basic Scenario

The next adversary represents a common scenario for malware analysts: incident response. This refers to the situation in which an analyst is called to assess the damage achieved by a piece of malware that has already infected a computer and currently still running on it [CMGS12]. In this scenario, the targeted computer is part of critical infrastructure which cannot be taken offline: e.g., a power control system. The analyst does not have an image of the computer that contains the uninfected state and must perform analysis on the infected image without being detected by the malware. That is, the analyst has access to E where M has already been loaded. However, they can gain no information about E without loading themselves, which changes E.

Basic Soundness We define the *basic soundness* game as follows:

Experiment $\mathbf{Exp}^{\mathrm{sou}}_{M,E,A,\mathtt{Load}}$:
$(\mathcal{D}_1, S_1, ..., \mathcal{D}_n, S_n) \leftarrow \mathcal{M}(E)$
$E_M \leftarrow \mathtt{Load}(M, E)$
$E_{M,A} \leftarrow \mathtt{Load}(A, E_M)$
For $i = 1$ to n
  $\mathtt{Guess}_i = A^{D_i(\cdot)}(E_{M,A}, S_i, D_{i-1}, S_{i-1}, ..., D_1, S_1)$.
If $\forall i$,$\mathcal{D}_i(\mathtt{Guess}_i) = 1$
  return 1
Else
  return 0.

Definition 8. *Let* Load *be a program loading module. A program M is ϵ-sound for the target E (drawn from $\mathcal{E}$) with respect to A if*

$$\Pr[\mathbf{Exp}^{\mathrm{sou}}_{M,E,A,\mathtt{Load}}(\cdot) = 1] > 1 - \epsilon.$$

Our results in this model are slightly more complicated than those in the Blind game. By our earlier-stated assumption, the analyst loading their tools causes some change in E. For the malware to successfully evade, this change must be large enough such that the analyst cannot easily guess values that will make D_i

accept. If the analyst only overwrites a few bits, for example, they can trivially guess the correct sequence. We will formalize this notion, noting where our model differs from reality.

First, we assume for convenience that loading A changes a random subset of locations of size ν. This differs somewhat from reality, in which changes will be limited to certain subsets of the environment (such as the filesystem or registry). However, in both cases, portions of E that would not be overwritten by A can be ignored by both M and A.[5] We further assume that the locations of $E_{M,A}$ that are changed are known to A and they are set to values independent of the values in E_M. We also assume that the M is always able to execute with A loaded. This requires that loading A never overwrites M's functionality; in practice, analysts avoid overwriting the program they are analyzing, so this assumption holds.

Theorem 4. *Let M be a (β, γ)-authenticating piece of malware with n decision procedures and maximum read size α where $n \cdot \alpha = \ell^c$ for some $0 < c < 1$. Furthermore, suppose that M is δ correct on all $E_M \leftarrow \texttt{Load}(M, E)$. Let $c' > 0$ be some parameter. If there exists some A with artifact size $\nu = \ell^{1-c}$, then by making at most $2^{c'+2}$ oracle queries M is at most $(e^{-1/4\ell^{1-c}} + \delta + e^{-2/3c'^2})$-basic sound.*

The proof of this theorem can be found in Appendix A.3. Roughly, when the product of the read size of the malware and the size of the analyst is at most the total size of the environment ℓ we expect the malware read locations and the analyst to collide in a small (logarithmic) number of positions. The analyst is then able to exhaustively search over the relevant locations that were erased. We simplify the theorem for common parameter settings:

Corollary 3. *Let λ be a security parameter where $\ell = \texttt{poly}(\lambda)$. Let M be a (β, γ)-authenticating piece of malware with n decision procedures and maximum read size α where $n \cdot \alpha = \ell^c$ for some $0 < c < 1$. Furthermore, suppose that M is δ correct on all $E_M \leftarrow \texttt{Load}(M, E)$. If there exists some A with artifact size $\nu = O(\ell^{1-c})$, then by making at most $\texttt{poly}(\lambda)$ oracle queries M is at most $(\delta + 1/\texttt{poly}(\lambda))$-basic sound.*

The above statement says that if the product of the size of the sensed positions and the analyst size is less than the total environment length then it is possible for the A to evade the malware and force the decision procedures to output 1.

We now proceed to show a sufficient condition for security. The necessary condition requires that the intersection between S_i and A is large. However, it also requires that A is not able to come up with valid guesses for the missing parts of the E. Creating a simple definition for this condition is complicated by two factors:

[5] In reality, we expect certain portions of E to be more likely to be overwritten by different A. Our results extend to that model.

1. The malware, M, does not know ahead of time where A will be loaded. If A can load in a location S_i whose values, E_{S_i} are easy to predict, it is impossible for M to provide security.
2. Once loaded, the A has access to the rest of E. This means that any redundancies or observable patterns or structures in E can be used to increase A's probability of guessing successfully.

Combining these two requirements, M should sense from as much of the environment as possible and E at sensed locations has to be hard to predict even knowing the rest of the environment. It is unlikely that computer systems satisfy these requirements. Environments have known structures and patterns – OS structures, filesystem contents, common libraries, etc. – and there are areas that have very low entropy. To codify the difficulty of satisfying these requirements, we present an analogue of Definition 3 and a corresponding sufficient condition for security. However, our condition should be seen as a largely negative result, as it only applies under unrealistic conditions on E and M.

Definition 9. *Let λ be a security parameter. A piece of M is μ-entropic sensing if for every subset $E_{sub} \subset E$ such that $|E_{sub}| \geq \mu$, then*

$$\min_{1 \leq i \leq n} \left(\max_{E' \in \mathcal{E}} \left(\Pr_{\substack{E \leftarrow \mathcal{E} \wedge \mathcal{D}_i, \\ S_i \leftarrow M(E)}} [\mathcal{D}_i(E'_{S_i}) = 1 | \mathcal{D}_1, S_1, ..., \mathcal{D}_{i-1}, S_{i-1}, E \setminus E_{sub}] \right) \right) = \mathtt{negl}(\lambda)$$

where $\mathcal{D}_i$ is the entire truth table of $\mathcal{D}_i$ and $E \setminus E_{sub}$ is the portion of E which is not contained in E_{sub}.

Definition 9 imposes a constraint both on the malware and on the environmental distribution $\mathcal{E}$ itself. This implicitly requires that all large subsets of $\mathcal{E}$ have super-logarithmic min-entropy conditioned on the rest of the environment.

Theorem 5. *Let λ be a security parameter. Let M be a μ-entropic sensing with n decision procedures. If all A have artifact size at least μ, then any black-box A making at most $\mathtt{poly}(\lambda)$ oracle queries then M is at least $(1 - \mathtt{negl}(\lambda))$-basic sound.*

The proof of this theorem can be found in Appendix A.4. Most of the complexity of the proof is contained in Definition 9 which implies that the analyst's first guess on some decision procedure succeeds with negligible probability. Standard arguments show that even with a polynomial number of guesses their overall success remains negligible.

Note: It is possible to weaken Definition 9 to be probabilistic. That is, there is a good chance that the set overwritten by the A will make it difficult to provide good inputs to some $\mathcal{D}_{i^*}$. However, this does not fundamentally change the character of the result which says that all large subsets of E must be entropic and that M must read all subsets of E with good probability.

5 Resettable Adversary

Finally, we turn to our least setting which we call the *resettable* adversary. In this setting A is allowed access to the malware M and the environment E while they are still separated. They are allowed to `Load` in the environment E multiple times and reset. Not surprisingly, our results in this model are negative. As long as there are multiple analysis techniques that are disjoint it is always possible for the analyst to acquire the state of the environment that exists without the analyst being present. This allows the analyst to present the pristine environment to the malware, thus unlocking it. We begin by formalizing the interaction.

Resettable Soundness We define the *resettable soundness* game as follows:

Experiment $\mathbf{Exp}^{\text{res}-\text{sou}}_{M,E,A,\texttt{Load},\psi}(\cdot)$:
$(\mathcal{D}_1, S_1, ..., \mathcal{D}_n, S_n) \leftarrow M(E)$
$E_M \leftarrow \texttt{Load}(M, E)$
$state = \perp$
For $i = 1$ to ψ
  $A_i \leftarrow A(\mathcal{D}_1, S_1, ..., D_{i-1}, S_{i-1}, D_i, S_i, state)$
  $E_i \leftarrow \texttt{Load}(A_i, E)$
  $(\texttt{Guess}_i, f, state) = A_i(E_i, \mathcal{D}_i, S_i)$.
  if $f = 1$ break
If $\forall i$, $D_i(E_i) = 1$ return 1
Else return 0.

Definition 10. *Let* `Load` *be a program loading module and let* λ *be a security parameter. A program* M *is* ϵ*-resettable sound for the target* E *(drawn from* $\mathcal{E}$*) with respect to* A *if for all* $\psi = \texttt{poly}(\lambda)$*,*

$$\Pr[\mathbf{Exp}^{\text{sou}}_{M,E,A,\texttt{Load},\psi}(\cdot) = 1] > 1 - \epsilon.$$

Theorem 6. *If there exists multiple analysis techniques* A_1, A_2 *such that the locations overwritten by* A_1, A_2 *are disjoint, then all* M *that is* δ*-correct is at most* δ*-resettable sound.*

Proof (Sketch). The analyst A proceeds in three stages. First, they load some tool A_1 and output all non overwritten parts of E as state. They then load A_2 that does the same. Finally, they create an A_3 that encodes a copy of the entire environment as it exists without any tools present. This A_3 recreates the proper inputs to the decision procedures and only fails when M fails to authenticate in the legitimate environment.

In the above proof sketch we assume that A_3 is able to encode the entire target environment E in an analysis module. In reality, once the analyst has recovered the environment, they can produce an module that only includes the relevant information which is read in by M. The only requirement for the analyst is to be able to encode the entire environment and their guessing logic on the

target machine. One could imagine that the loaded module could communicate with outside storage for pieces of the environment but this is out of scope for our model.

The resettable analyst A can forge the pristine environment and thus unlock the malware. With the ability to reset the environment and malware, the analyst can understand the entire target environment with the same precision as the malware making security impossible.

A Proofs

A.1 Proof of Theorem 2

Proof (Proof of Theorem 2). We show a stronger statement, we show a single algorithm A that works for any (β, γ)-environmentally authenticating malware. Let $t = 2^\beta \ln\left(\frac{n}{1-\epsilon}\right)$. Define A as follows for decision procedure i:

1. Input $\mathcal{D}_i, S_i, \mathcal{D}_{i-1}, S_{i-1}, ..., \mathcal{D}_1, S_1$.
2. For $j = 1$ to t
   (a) Sample $E_j \leftarrow \mathtt{Sam}_{\mathcal{E}}$.
   (b) If $D_i(E_{j,S_i}) = 1$ output $\mathtt{Guess}_i = E_j$.
3. Output $\perp$.

This procedure is repeated for each decision procedure. A wins if all decision procedures output 1. We first note that the probability that some decision procedure is incorrect is bounded by at most δ. We now bound the probability that A outputs $\perp$ for any iteration conditioned on the malware being correct. We first consider a single iteration. By Definition 4 and Assumption 1, $\mathbb{E}_{E_j \in \mathcal{E}}(\Pr[D_i(E_{j,S_i}) = 1]) \geq 2^{-\beta}$. That means that

$$\begin{aligned}
\Pr[A \text{ outputs } \perp \text{ on } D_i] &= \forall j, \Pr[D_i(E_{j,S_i}) = 0] \\
&= \left(\mathbb{E}_{E \in \mathcal{E}} \Pr[D_i(E_{S_i}) = 0]\right)^t \\
&= \left(1 - \mathtt{Accept}(M, i, \mathcal{E})\right)^t \\
&\leq \left(1 - 2^{-\beta}\right)^t \\
&\leq \left(\left(1 - 2^{-\beta}\right)^{2^\beta}\right)^{\left(t/2^\beta\right)} \\
&\leq \left(\frac{1}{e}\right)^{\left(\frac{t}{2^\beta}\right)} \leq e^{-t/2^\beta}. \qquad (1)
\end{aligned}$$

Then across all iterations by union bound and Equation 1: $\Pr[A \text{ outputs } \perp \text{ on any } \mathcal{D}_i] \leq ne^{-t/2^\beta}$. That is,

$$\Pr[\mathbf{Exp}^{\text{bli-sou}}_{M,\mathcal{E},A}(\cdot) = 1] \geq 1 - ne^{-t/2^\beta} = 1 - ne^{-\ln(n/(1-\epsilon))} = 1 - n\left(\frac{1-\epsilon}{n}\right) = \epsilon.$$

Note that the overall running time of A is at most $t_A = n(t_E + t_{oracle}) \cdot t$ as required. The statement of the theorem is achieved by adding the probability δ that the malware is incorrect.

A.2 Proof of Theorem 3

Proof (Proof of Theorem 3). Let A be a black box algorithm that only provide inputs to the current decision algorithm. Since the entire decision procedure is revealed once a "true" input is found there is no reason to query a previous decision algorithm. Consider some decision algorithm i^* that minimizes the probability in Definition 3. We bound the probability that A can make $\mathcal{D}_{i^*}$ output 1 as this bounds the probability of all algorithms outputting 1 (it may be that only a single decision algorithm outputs 0 on some inputs). The only information about values E that cause $\mathcal{D}_i$ to output 1 are contained in the query responses. Since the adversary wins if they get a single 1 response we can assume that A makes t deterministic queries and if none of those responses is 1 their guess will also be a deterministic value. Denote by $g_1, ..., g_{t+1}$ these values. Then we bound:

$$\begin{aligned}
&\sum_{j=1}^{t+1} \Pr_{D_i, S_i \leftarrow M}[D_{i^*}(g_j) = 1] \leq \Pr[D_{i^*}(g_1) = 1] + \Pr[D_{i^*}(g_2) = 1 | g_1 = 0] + ... \\
&+ \Pr[D_{i^*}(g_{t+1}) = 1 | D_{i^*}(g_1) = 0 \wedge ... \wedge D_{i^*}(g_t) = 0] \\
&\leq 2^{-\gamma} + \frac{\Pr[D_{i^*}(g_2) = 1 \wedge D_{i^*}(g_1) = 0]]}{\Pr[D_{i^*}(g_1) = 0]} + ... \\
&+ \frac{\Pr[D_{i^*}(g_{t+1}) = 1 \wedge D_{i^*}(g_1) = 0 \wedge ... \wedge D_{i^*}(g_t) = 0]}{\Pr[D_{i^*}(g_1) = 0 \wedge ... \wedge D_{i^*}(g_t) = 0]} \\
&\leq 2^{-\gamma} + \frac{\Pr[D_{i^*}(g_2) = 1]}{\Pr[D_{i^*}(g_1) = 0]} + ... + \frac{\Pr[D_{i^*}(g_{t+1}) = 1]}{\Pr[D_{i^*}(g_1) = 0 \wedge ... \wedge D_{i^*}(g_t) = 0]} \\
&\leq 2^{-\gamma} + \frac{\Pr[D_{i^*}(g_2) = 1]}{1 - 2^{-\gamma}} + ... + \frac{\Pr[D_{i^*}(g_{t+1}) = 1]}{1 - t2^{-\gamma}} \\
&\leq (t+1)\frac{2^{-\gamma}}{1 - t2^{-\gamma}}
\end{aligned}$$

A.3 Proof of Theorem 4

Proof (Proof of Theorem 4). The adversary A does not know where in E that the malware M exists, A runs the risk of overwriting the sensors positions S_i. As stated above, we assume that M is operable after A has been loaded. The total size of M's reads from E are of size at most $n \cdot \alpha$. We define a single A that works for all M. Let A overwrite a random set of ν locations. However, rather than considering this A we instead consider some A' that overwrites each element of E_M with probability $2\nu/\ell$. Note that,

$$\Pr[||A'| < \nu] = \Pr\left[|A'| < (1 - \frac{1}{2})\mathbb{E}|A'|\right] = e^{-1/8\mathbb{E}|A'|} = e^{-1/4\nu} = e^{-1/4\ell^{1-c}}$$

using the multiplicative version of the Chernoff bound. Assume that A' simply outputs $\perp$ in this setting. Thus, all of A' success occurs when it overwrites at least ν positions and the job of A' to provide inputs to $\mathcal{D}_i$ is at least as difficult as A. For the reminder of the proof we consider A'.

We now bound the size of the intersection between the locations read by M and the locations overwritten by $\texttt{Load}(A', E_M)$. Denote by E_{bad} the locations overwritten by $\texttt{Load}(A', E_M)$ conditioned on the event that A' overwrites at least ν locations.

To bound the success probability of A', we care about the size of the intersection between the locations read by M and overwritten by E_{bad}. Since E_{bad} represents ν random locations the intersection between $(\cup_i S_i) \cap E_{bad}$ is distributed as a Binomial distribution, which we denote as X, with parameters $B(n\alpha, 2\nu/\ell)$. Then one has that,

$$\mathbb{E}[X] = \frac{2\nu n\alpha}{\ell} = \frac{2\ell^c \ell^{1-c}}{\ell} = 2.$$

Let $c' > 0$ be a constant. By a second application of the Chernoff bound one has that:

$$\Pr[X > 2 + c'] = e^{-2/3c'^2}.$$

For an intersection of size κ the correct E_M can be found using 2^κ oracle queries. Note that this is an upper bound, in the setting where a decision algorithm takes a smaller number of corrupted bits, these bits can be recovered in parts. Here we assume that all corrupted bits are necessary for a single decision algorithm. The statement of the theorem follows by using an A' that exhaustively searches over corrupted bits when the size of the corrupted bits is at most $c' + 2$ and aborts otherwise.

A.4 Proof of Theorem 5

Proof (Proof of Theorem 5). Consider some A with artifact size at least μ. Let A be a black box algorithm that only provide inputs to the current decision algorithm. Since the entire decision procedure is revealed once a "true" input is found there is no reason to query a previous decision algorithm. Denote by E_{sub} the subset of size at least μ that is overwritten by $\texttt{Load}(A, E_M)$, Then by Definition 9. There exists some i^* such that

$$\left(\max_{E' \in \mathcal{E}} \left(\Pr_{E \leftarrow \mathcal{E} \wedge \mathcal{D}_i, S_i \leftarrow M(E)} [\mathcal{D}_i(E'_{S_{i^*}}) = 1 | \mathcal{D}_1, ..., \mathcal{D}_{i^*-1}, S_1, ..., S_{i^*-1}, E \setminus E_{sub}] \right)\right) = \texttt{negl}(\lambda).$$

We bound the probability that A can make $\mathcal{D}_{i^*}$ output 1 as this bounds the probability of all algorithms outputting 1 (it may be that only a single decision algorithm outputs 0 some fraction of the time). The only information about values E that cause $\mathcal{D}_{i^*}$ to output 1 are contained in the query responses. Since

the adversary wins if they get a single 1 response we can assume that A makes $t = \texttt{poly}(\lambda)$ deterministic queries and if none of those responses is 1 their guess will also be a deterministic value. Denote by $g_1, ..., g_{t+1}$ these values. Then we bound:

$$\sum_{j=1}^{t+1} \Pr_{D_i, S_i \leftarrow M}[D_{i^*}(g_j) = 1] \leq \Pr[D_{i^*}(g_1) = 1] + \Pr[D_{i^*}(g_2) = 1 | g_1 = 0] + ...$$

$$+ \Pr[D_{i^*}(g_{t+1}) = 1 | D_{i^*}(g_1) = 0 \wedge ... \wedge D_{i^*}(g_t) = 0]$$

$$\leq \texttt{negl}(\lambda) + \frac{\Pr[D_{i^*}(g_2) = 1 \wedge D_{i^*}(g_1) = 0]]}{\Pr[D_{i^*}(g_1) = 0]} + ...$$

$$+ \frac{\Pr[D_{i^*}(g_{t+1}) = 1 \wedge D_{i^*}(g_1) = 0 \wedge ... \wedge D_{i^*}(g_t) = 0]}{\Pr[D_{i^*}(g_1) = 0 \wedge ... \wedge D_{i^*}(g_t) = 0]}$$

$$\leq \texttt{negl}(\lambda) + \frac{\Pr[D_{i^*}(g_2) = 1]}{\Pr[D_{i^*}(g_1) = 0]} + ... + \frac{\Pr[D_{i^*}(g_{t+1}) = 1]}{\Pr[D_{i^*}(g_1) = 0 \wedge ... \wedge D_{i^*}(g_t) = 0]}$$

$$\leq \texttt{negl}(\lambda) + \frac{\Pr[D_{i^*}(g_2) = 1]}{1 - \texttt{negl}(\lambda)} + ... + \frac{\Pr[D_{i^*}(g_{t+1}) = 1]}{1 - t\texttt{negl}(\lambda)}$$

$$\leq (t+1)\frac{\texttt{negl}(\lambda)}{1 - t\texttt{negl}(\lambda)} = \texttt{negl}(\lambda)$$

References

[Bau14] Car Bauer. ReMASTering Applications by Obfuscating during Compilation. blog post, August 2014.

[BCK+10a] Davide Balzarotti, Marco Cova, Christoph Karlberger, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. Efficient detection of split personalities in malware. In *NDSS*. Citeseer, 2010.

[BCK+10b] Davide Balzarotti, Marco Cova, Christoph Karlberger, Christopher Kruegel, Engin Kirda, and Giovanni Vigna. Efficient Detection of Split Personalities in Malware. In *In Proc. of the Symposium on Network and Distributed System Security (NDSS)*, 2010.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (Im)possibility of Obfuscating Programs. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, pages 1–18, London, UK, UK, 2001. Springer-Verlag.

[BKY16] Jeremy Blackthorne, Benjamin Kaiser, and Bülent Yener. A formal framework for environmentally sensitive malware. In *Research in Attacks, Intrusions, and Defenses - 19th International Symposium, RAID 2016, Paris, France, September 19-21, 2016, Proceedings*, pages 211–229, 2016.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.

[CAM+08] Xu Chen, J. Andersen, Z.M. Mao, M. Bailey, and Jose Nazario. Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In *Dependable Systems and Networks With FTCS and DCC,*

2008. DSN 2008. IEEE International Conference on, pages 177–186, June 2008.

[CMGS12] Paul Cichonski, Tom Millar, Tim Grance, and Karen Scarfone. Computer Security Incident Handling Guide: Recommendations of the National Institute of Standards and Technology, 800-61. Revision 2. *NIST Special Publication*, 800-61:79, 2012.

[CTL97] Christian Collberg, Clark Thomborson, and Douglas Low. A Taxonomy of Obfuscating Transformations, 1997.

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 523–540. Springer, 2004.

[DRS08] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors–a brief survey of results from 2004 to 2006. In *Security with Noisy Data*. Citeseer, 2008.

[DRSL08] Artem Dinaburg, Paul Royal, Monirul Sharif, and Wenke Lee. Ether: malware analysis via hardware virtualization extensions. In *CCS '08 Proceedings of the 15th ACM conference on Computer and communications security*, pages 51–62, 2008.

[Fer07] Peter Ferrie. Attacks on More Virtual Machine Emulators. Technical report, Symantec Advanced Threat Research, 2007.

[Fer11] Peter Ferrie. The Ultimate Anti-Debugging Reference, May 2011. [Online]. Available: http://pferrie.host22.com/papers/antidebug.pdf. Accessed Apr. 6, 2015.

[FKSW06] Ariel Futoransky, Emiliano Kargieman, Carlos Sarraute, and Ariel Waissbein. Foundations and applications for secure triggers. In *In ACM Transactions of Information Systems Security*, page 2006, 2006.

[FRS16] Benjamin Fuller, Leonid Reyzin, and Adam Smith. When are fuzzy extractors possible? In *Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I 22*, pages 277–306. Springer, 2016.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova 0001, Amit Sahai, and Brent Waters. Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits. In *FOCS*, pages 40–49. IEEE Computer Society, 2013.

[ICF+15] Gene Itkis, Venkat Chandar, Benjamin W Fuller, Joseph P Campbell, and Robert K Cunningham. Iris biometric security challenges and possible solutions: For your eyes only? using the iris as a key. *IEEE Signal Processing Magazine*, 32(5):42–53, 2015.

[Jst16] Jsteube. oclGaussCrack, 2016.

[KLZS12] Clemens Kolbitsch, Benjamin Livshits, Benjamin Zorn, and Christian Seifert. Rozzle: De-cloaking Internet Malware. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 443–457, Washington, DC, USA, 2012. IEEE Computer Society.

[Kra10] Hugo Krawczyk. Cryptographic Extraction and Key Derivation: The HKDF Scheme. In *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference*, volume 6223 of *Lecture Notes in Computer Science*, pages 631–648. Springer, 2010.

[KYH+09] Min Gyung Kang, Heng Yin, Steve Hanna, Stephen McCamant, and Dawn Song. Emulating emulation-resistant malware. In *Proceedings of the 1st ACM workshop on Virtual machine security*, VMSec '09, pages 11–22, New York, NY, USA, 2009. ACM.

[LKMC11] Martina Lindorfer, Clemens Kolbitsch, and Paolo Milani Comparetti. Detecting Environment-sensitive Malware. In *Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection*, RAID'11, pages 338–357, Berlin, Heidelberg, 2011. Springer-Verlag.

[Moo15] Paul Moon. The Use of Packers, Obfuscators and Encryptors in Modern Malware. Technical report, Information Security Group, Royal Holloway University of London, 2015.

[NTS99] Noam Nisan and Amnon Ta-Shma. Extracting randomness: A survey and new constructions. *Journal of Computer and System Sciences*, 58(1):148–173, 1999.

[NZ96] Noam Nisan and David Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–52, 1996.

[PMRB09] Roberto Paleari, Lorenzo Martignoni, Giampaolo Fresi Roglia, and Danilo Bruschi. A Fistful of Red-pills: How to Automatically Generate Procedures to Detect CPU Emulators. In *Proceedings of the 3rd USENIX Conference on Offensive Technologies*, WOOT'09, pages 2–2, Berkeley, CA, USA, 2009. USENIX Association.

[Rén61] Alfréd Rényi. On measures of entropy and information. In *Proceedings of the fourth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 547–561, 1961.

[RS98] James Riordan and Bruce Schneier. Environmental Key Generation Towards Clueless Agents. In *Mobile Agents and Security*, pages 15–24, London, UK, UK, 1998. Springer-Verlag.

[RT12] Kaspersky Lab Global Research and Analysis Team. Gauss: Abnormal Distribution. Technical report, Kaspersky Lab, 2012.

[SH12] Michael Sikorski and Andrew Honig. *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, San Francisco, CA, USA, 1st edition, 2012.

[SRL12] Chengyu Song, Paul Royal, and Wenke Lee. Impeding Automated Malware Analysis with Environment-sensitive Malware. In *Hotsec*, 2012.

[SWP08] Amitabh Saxena, Brecht Wyseur, and Bart Preneel. White-box cryptography: Formal notions and (im) possibility results. 2008.

[XZGL14] Zhaoyan Xu, Jialong Zhang, Guofei Gu, and Zhiqiang Lin. GOLDENEYE: Efficiently and Effectively Unveiling Malware's Targeted Environment. In *Research in Attacks, Intrusions and Defenses*, pages 22–45. Springer, 2014.

[Yan13] Lok Kwong Yan. Transparent and precise malware analysis using virtualization: from theory to practice. 2013.