

Optimising Malware

José M. Fernandez
École Polytechnique de Montréal
2500 chemin de Polytechnique
Montréal, Québec, Canada
jose.fernandez@polymtl.ca

Pierre-Marc Bureau
École Polytechnique de Montréal
2500 chemin de Polytechnique
Montréal, Québec, Canada
pierre-marc.bureau@polymtl.ca

ABSTRACT

In recent years, malicious software (malware) has become one of the most insidious threats in computer security. However, this is arguably not the result of increased sophistication in malware design or attack strategies, but rather of the increased presence of computers and computer networks within every aspect of society. In this paper, we address and defend the commonly shared point of view that the worst is very much yet to come. We introduce an aim-oriented performance theory for malware and malware attacks, within which we identify some of the performance criteria for measuring their “goodness” with respect to some of the typical objectives for which they are currently used. We also use the OODA loop model, a well-known paradigm of command and control borrowed from military doctrine, as a tool for organising and reasoning about the behavioural characteristics of malware and orchestrated attacks using it. We then identify and discuss particular areas of malware design and deployment strategy in which very little development has been seen in the past, and that are likely sources of increased future malware threats. Finally, we discuss how standard optimisation techniques could be applied to malware design, in order to allow even moderately equipped malicious attackers to quickly converge towards optimal malware attack strategies and tools fine-tuned for the current Internet.

Categories and Subject Descriptors

D.4.6 [Operating Systems]: Security and Protection—*Invasive Software*; K.6.5 [Management of Computer and Information Systems]: Security and Protection—*Invasive Software, Unauthorized Access*

General Terms

Design, Performance, Security, Theory

Keywords

Malicious software, malware, performance, optimisation

1. INTRODUCTION AND BACKGROUND

Cría cuervos y te comerán los ojos ...
— Spanish proverb

The term *malware*, a derivation from *malicious software*, has been coined to describe a broad category of software tools that have been programmed for malicious purposes.

While the use of malware and the negative impact it is having on our society is on the rise, it is debatable whether this is a consequence of a significant increase in sophistication in malware technology. In fact, malware has been present in computers for a very long time. What can be learned from its history? For one, that writing good and completely autonomous malware, especially worms, is probably very hard... This is probably one of the reasons why the most common, and probably the most successful, network attack strategies combine automated network reconnaissance with the limited use of attack vectors on selected targeted machines, in order to construct large botnets [6, 8]. Second, that most significant developments have been concentrated on penetration, propagation and detection avoidance.

As far as penetration is concerned, the large numbers of available attack vectors [13] used in automated and directed malware attacks can hardly be attributed to significant advances in malware technology. Most of these vectors re-use the same handful of software vulnerability classes and system misconfiguration errors. The variety of available vectors is simply a function of the variety of software of increasing complexity providing services on networked systems, and thus providing an ever increasing number of opportunities for undetected or unpatched vulnerabilities and misconfiguration errors to be exploited. The increased virulence and propagation ratios observed in recent worm epidemics is due mostly to the large number of machines and the high level of instant interconnectivity provided by the Internet. In fact, safe for some theoretical models of fast-spreading worms (which we will discuss in Section 3), the propagation algorithms observed in in-the-wild worms and other forms of malware is relatively unsophisticated, when not flawed...

As for detection avoidance, there is little new under the sun. While there has been a recent recrudescence of techniques such as code polymorphism, code hiding and anti-virus detection and neutralisation, the first appearance of these features dates back to the days of boot-sector and executable file viruses. There are however, two interesting novelties in this area: the detection of virtual environments and the use of covert channels in combination with backdoors. While these provide an interesting example of co-evolution [10] between malware and malware counter-measures (honeypots or manual malware code reverse engineering in the former, and intrusion detectors in the latter), we believe that such innovations, like most others in malware technology, belong more in the “careful craft” rather than in the “significant paradigm shift” category. In other words, there has been so far little science and engineering in malware design.

Today, this is somewhat surprising, especially given the fact that other sectors of malicious activity on the Internet, such as spam, have shown increasing levels of sophistication, to the point that some of the techniques used today were the object of academic research barely a few years ago.

The most notable exception to this, we believe, is the introduction of backdoors and their use as control tools for coordinated malware attacks, and the subsequent introduction of botnets, which by automating the process make it possible to effect control on significantly larger structures. This significant change immediately suggests a natural connection between the planning and execution of malware attacks and considerations of military doctrine, on the one hand, and resource optimisation (such as in operations research), on the other. The objectives and preoccupations arising from these fields, naturally inspire questions about malware attacks like: What would be the most effective strategy for effectively constructing, using and maintaining botnets for a given pre-determined purpose? On the technological level, this question becomes the following: what new malware coding techniques or algorithms would most effectively serve this higher purpose?

Addressing these questions has been the motivation of our research. Our objective is to understand the malware conception and optimisation process in order to build powerful and scalable defence infrastructures. In this paper, we introduce in Section 2 a basic performance theory for malware and malware-based attacks within which to formalise what it means for malware to be “good”. We then describe and discuss in Section 3 the performance criteria that are the most likely indicators of success for the various types of objectives that malware is currently used for. We use the OODA loop model in Section 4 to organise and classify the various aspects of malware attacks strategy and malware design that constitute the decision and design space of malware construction. We then identify in Section 5 what aspects of malware strategy and design remain mostly unexplored. We further discuss how standard optimisation techniques could be used to construct the malware threats of tomorrow. We close in Section 6 with a summary of our results, main conclusions, and directions of future research.

2. BASIC CONCEPTS AND PERFORMANCE MODEL

We use the term *malware attack* in a broad sense, to refer to any action performed by a malicious entity with the help of malicious software in order to achieve a precise purpose. In such attacks, the malicious attacker may use different kinds of malware and execute several concurrent instances of it on different machines. Malware instances may only play a small part in the general execution of the attack. For example, a malicious hacker penetrating a system to steal valuables by obtaining passwords by social engineering, or even by using an exploit tool to take advantage of a software vulnerability, does not constitute a malware attack *per se*. On the other hand, an attack with the same purpose using Trojan horses distributed by email to obtain passwords or to plant backdoors, would constitute a malware attack, even if malware was not involved in all phases of the attack.

If we compare a computer to a battlefield and a computer network or large computer system to a region or country, then a malware attack is akin to conducting an offensive

campaign against the targeted systems. The malware tools used in that campaign are the tactical units that are used to fight specific battles or to perform specific tasks throughout the campaign. To use common military terminology, questions about how a campaign is conducted are called *strategic* and questions about how the individual battles are fought, are called *tactical*. In our case, we will use the term *strategy* to refer to the design and execution of the overall attack, which might be implemented or controlled directly by the malicious attacker or software he is using. The techniques used in the construction of malware and the tasks executed by them on the targeted systems, we will refer to as *tactics*.

Let us now turn to the question of performance of malware and malware attacks. Malware can be seen as intrusion agents built to help their creator reach their objectives. A given type of malware will have been programmed with particular characteristics or behavioural traits, that are parametrised by an associated set of variables. Each instance of this type of malware will behave differently, depending on the particular values of these parameters, whether these are fixed by design or are dynamically set as a result of interaction with their environment. For example, some of the characteristics of a malicious software include the algorithm it uses for propagation and reproduction and the penetration methods, while the associated parameters might include speed of propagation, the particular exploit(s) used to penetrate the targets, etc. Because these malware instances will operate under different environments, it is to be expected that their performance will be affected.

The same pattern can be applied to malware attacks. A malware attack might be planned and conducted according to a particular strategy (chosen within a strategy “play book” or strategic design space), defined by a set of characteristics or traits that they include (e.g. a reconnaissance phase). Each of these traits might in turn be parametrised by values (e.g. the range of IP addresses to be explored, the type of scanning, etc.).

The performance model we will use for malware attacks, and for malware agents performing tasks within, is based on the following principles:

1. “Performance” is not uniquely defined and several different performance criteria might exist and be relevant, depending on the objectives..

Some examples of performance criteria are detection ratio, infection coverage, and bandwidth utilised. The choice of criteria that are pertinent depend on the objectives of the malware attacker. For example, if an attacker conducts a malware attack to steal credit card numbers, the performance will be depend upon the total number of card numbers stolen and the preservation of his anonymity. On the other hand, if a Trojan horse is used, that is sent to potential victims to obtain their credit card numbers, the performance of that particular malware might be measured in terms of deception ratio. Finally, if a malicious attacker wants to construct a botnet to perform distributed denial of service (DDoS) attacks, the performance of the attack will be evaluated according to the number of hosts infected and their total available upstream bandwidth. We will discuss the various relevant performance criteria that are relevant to malware in Section 3.

2. *The performance criteria might be influenced by the characteristics present in the malware design or malware attack strategy, and the corresponding parameter values.*

For example, the type of scanning algorithm and the choice of exploits used by malware within the context of a given malware attack will affect the number of infected hosts, which is one of the many performance criteria that the attack might be evaluated by (see Section 4). This principle motivates in turn the principle of optimised malware design that is the basis of this paper (see Section 5). In essence, the hope of the malware designer and malware attack strategist is that by changing the characteristics of the malware and those of the attack strategy, a set of choices will be found that optimises the performance criteria that are most important to them.

3. *The performance criteria might be influenced by the characteristics of the operating environment within which the malware operates or on which the malware attack is conducted.*

The relevant characteristics of the environment might include, for example, the network topology, the operating systems and software installed on the targets and the defence posture of the network being attacked (i.e. presence of defensive software or hardware countermeasures, level of awareness and readiness of system administrators, etc.). In general, these environmental characteristics might include almost every aspect that the malware attacker cannot control during the attack.

Motivated by these principles, we put forth the thesis that the analysis of performance should always take into consideration these three aspects: 1) characteristics of the malware or malware attack, 2) the particular performance criterion being considered, and 3) the environment within which they are operating. Thus, typical statements about malware performance could take one of the forms below:

$$\text{Fixed } E, \text{ Fixed } P \Rightarrow P(C_1) \leq P(C_2) \quad (1)$$

$$\text{Fixed } C, \text{ Fixed } P \Rightarrow P(E_1) \leq P(E_2) \quad (2)$$

$$\text{Fixed } C \Rightarrow P_1(E) = f(P_2(E)) \quad (3)$$

$$\text{Fixed } E \Rightarrow P_1(C) = g(P_2(C)) \quad (4)$$

Equation 1 describes the situation where, for example, a malware programmer wants to optimise the choice of characteristics (C_1 or C_2) with respect to the required performance criterion P and when the environment characteristics E are fixed. Equation 2 describes the variation of a given performance criterion P of a malware (attack) with fixed characteristics C but under different operating environments E_1 and E_2 . Finally, Equations 3 and 4 represent trade-offs between two different performance criteria P_1 and P_2 . Equation 3 describes how different performance criteria of a given malware might interact as the operating environment changes; this is useful for malicious attackers trying to understand how different objectives might conflict as the characteristics of the target vary. On the other hand, understanding the trade-offs in Equation 4 might help “defenders” of a particular system (i.e. the operating environment E of the malware) understand how risk reduction with respect to

one threat (corresponding to a given malicious objective and associated performance criterion P_1), might interact with exposure to another threat, possibly with different objectives (and a different associated performance criterion P_2) as the strategy and tactics characteristics of the attack vary (represented by C).

In this paper, we will concentrate on Equation 1, in that we consider scenarios in which a malicious entity (person or organisation) having a pre-determined aim wants to optimise the characteristics and parameters of a malware attack in a fixed target environment (e.g. the Internet of today).

3. MALWARE PERFORMANCE ANALYSIS

In order to evaluate the performance of malware attacks, we need to establish performance metrics, identify which of them are most relevant and become performance *criteria*, and finally study their quantitative behaviour when malware characteristics and operating environment change.

3.1 Previous work on malware performance

Other researchers have already studied the performance of malware and malware attacks. Most of them have concentrated on the propagation rate metric. Propagation of malware is the easiest characteristic to study in malware behaviour. This task is made easy because malware tends to spread quickly and with a signature that differs from normal human usage of computer resources. Staniford *et al.* [17] have used the “classical epidemic model” to analyse the spread of the Code Red worm. Zou *et al.* [19] developed a general model for Internet worms called the “two-factor worm model”. This model considers the contagion of a worm and the human reaction to its propagation. Chen *et al.* [3] developed a discrete-time model that considers patching and cleaning effects on the propagation of a malicious software. Note that both of these works already consider the interaction between malware performance (in this case propagation rate) and environmental characteristics (human reaction, patching, cleaning, etc.), a reflection of our third principle. Furthermore, the respective findings could in principle be expressed in terms similar to those of Equation 2. Finally, Garetto *et al.* [5] have presented an analytical technique based on Markov Chains to capture the spreading characteristics of malware.

While this work is of high merit from a quantitative point of view, it is relatively limited in terms of the variety of criteria considered. In our case, we are more interested in being thorough in the qualitative identification of all (or most) pertinent criteria. Once the recurring pertinent criteria of interest have been identified, more solid quantitative results can then be obtained (possibly in the forms of Equations 1–4).

3.2 Aim-specific performance criteria

We base our analysis on the fact that performance is an indicator of success, and that since success depends on the initial aim, performance thus depends on the ulterior motivation behind a malware attack. There are various reasons that could motivate a programmer to create malicious software. For the purposes of our analysis, we have identified six main motivations to write malware: 1) fraud, 2) information theft, 3) sale of access to computing resources, 4) fame and glory, 5) destruction, and 6) information warfare. We do not consider this list necessarily complete nor permanent. It is based on today’s menace landscape and will

probably change in the future. What is important is that it allows us to identify and classify the essential performance criteria for malware.

Fraud. It is a deception for personal gain, and it aims toward obtaining money or property by false pretences. Cybercriminals are using malware to conduct fraud, there is no question about it. This aim is clearly getting more and more popular amongst malware programmers. In the case of malware, the primary objective is often to steal credit card number or bank account information in order to gather sums money. The performance analysis of malware intended for fraud will be impacted by the **amount of money** that can be illicitly acquired while preserving the anonymity of its author. To perform computer fraud, a piece of malware will often have to convince users to give private information, this feature will also make the difference between “good” and “better” malware, i.e. malware that is more *credible* and deceiving to the targeted user; the associated performance criterion is the malware **credibility** or **deception ratio**. Furthermore, the **anonymity** of the author must be considered because fraud is illegal in most countries. The **number of infected hosts** or **infection coverage** also has to be considered because it increases the chances of finding valuable data. Finally, **persistence** on the infected system will have positive influence because the longer a malware is executed on a host, the more chances it has to gather important information.

Information theft. While attacks with this objective as motivation have not been *widely* observed on the Internet, they are on the rise and they do pose serious threat to organisations present on it. The objective of this type of attack is to penetrate deeply in an organisation’s network to steal sensitive information. The author can then sell this information to a third party or use it to black mail the target organisation. Information that can be stolen include source code, blue prints, financial records, etc. The performance criteria that have to be considered when building malware for information theft are the **penetration ratio**, where the malware attempts to deeply penetrate into the network under siege. **Stealth** is also an advantage because the longer a network is infected without the knowledge of the administrators, the bigger the **amount of information** that can be stolen. The communication between malware nodes and their owner needs to be fast to easily transfer big amounts of data. Finally, the **location of infected hosts** on the network has to be considered because in most cases, sensitive information will be concentrated in a network segment and on very few hosts. In this case, infecting hundreds of workstation when sensitive information is stored on a central repository is not in the malware attacker’s advantage. Similarly to fraud, **anonymity** of the malware creator and **persistence** are factors that have to be considered.

Sale of access. A new trend in malware has been to create networks of infected systems, i.e. *botnets*, and to sell access to them for various purposes [6, 14]. Such botnets have been used in the past to send unsolicited e-mails, conduct DDoS attacks, and perform large scale computation. The performance of malware attacks aimed for access sale is impacted by the **total available upstream bandwidth** (specially in DDoS attacks) and the **security of access** to the infected

hosts. The available bandwidth can be estimated by summing the bandwidth available to each infected host. The security of access to nodes of the botnet is also important in this type of attack. A botnet that is available to everyone will be worth less than a secure botnet with restricted access. The **infection coverage** has to be high in order for the administrator to sell enough access to make an attractive profit. Since operating a botnet and selling illegitimate access to computer systems is illegal, **anonymity** of the malware author is also an important criterion.

Fame and glory. The possibility of becoming famous or gaining “bragging rights” within the underground hacker communities is often a motivation for programmers to create malware. For that purpose, the attacker might program his malware to maximise impact on the Internet and in the real world, often by seeking media coverage. Every aspect of the created malware is designed to expose the “talents” of its creator. What better way to showcase these talents than by penetrating well-protected and notorious targets, visible to all? Unlike other motivations, this one is particularly interesting because it is the only one where the author wants to be known. The creation of a malware being a crime in most countries, its designer does not the authorities to find his identity, yet, the creator wants to be known and his talents to be exposed. This is often achieved by the use of a “signature” using a pseudonym. Finally, a malware attack that is only a variation of known attacks will not necessarily raise public attention, while a brand new epidemics showing new traits will. Thus, in this case the performance criteria would include the **location** and **infection coverage**, potential **damage**, and most of all, **originality**.

Destruction. In terms of numbers, most malware attacks on the Internet have been aimed towards destruction, both targeted (DDoS attacks, web site defacements) and non-targeted (paralysing viruses and worms). In addition, some malware observed in the wild, even without active payloads, have caused serious damage to network infrastructures due to their mere propagation traffic [9]. In certain situations, it is possible to damage physical components of a computer system from a software, for example by changing settings in BIOS memory from the operating system. If, furthermore, the affected computer systems are connected to and control critical infrastructure systems, such as railway traffic lights, oil refineries, electric power grid, etc., damage in hardware would not be necessarily limited to the infected computer systems themselves. The associated performance criterion would be the total amount of **hardware and material damage** inflicted as a direct or indirect consequence of the malware attack. Of course, the **location of infected hosts** will also change the impact of the destruction. The **infection coverage** is both a metric of non-permanent damage and an enabler to induce greater permanent damages. The same is true for the **total upstream bandwidth** of the infected network, which will increase DDoS-attack capability. The **speed of propagation** or **propagation rate**, will impact how much damage an epidemics can do, because it will diminish the effectiveness of defensive counter-measures. Finally, in certain cases **anonymity** of the malware attackers will be of great advantage for that same reason, as it will delay counter-attack efforts.

Information warfare. It is the extension of the battlefield to information technology. In doctrinal terms, it is the offensive and defensive use of information and information systems to deny, exploit, corrupt, or destroy, an adversary’s information or information system. It targets government or economic infrastructures and military communication channels. The first and most obvious criterion is the amount of **hardware and material damage**. In some cases, however, it might not be necessary and it might even be preferable not to permanently destroy targets, but only to temporarily *disrupt* them, i.e. performing only “soft kills” of these systems; the associated criterion is the total length (in time) of **system disruption** or **unavailability**. The **speed of propagation** is also important, as it lowers the chances of effective reaction by the defender. The **location of infected hosts** will impact the amount of damage inflicted, but also the effectiveness of the attack on other non-material levels. For example, the information operations and psychological warfare aspects of an information warfare campaign will be affected by the overall **disinformation exposure** of targeted web sites that have been defaced or altered (i.e. number of client stations that will surf them).

In summary, we have identified various aim-specific performance criteria, some of which are common to two or more of the broad categories of motivation we have described; they are shown in Table 1. Thus, it might be useful to divide these performance criteria into two broad categories: *generic performance criteria*, that are applicable to all or almost all of the above objectives, and *specific performance criteria*, that are applicable to one, or only a few, of the above motivations. However, we do not claim that this list of criteria is necessarily exhaustive. It does however contain what we believe are some of the most relevant criteria that must be considered when evaluating the performance of malware attacks. What the “ultimate” list of performance criteria should be, only the malware attacker really knows...

4. CHARACTERISATION OF MALWARE ATTACKS

In order to analyse the performance of malicious software, we need to describe its characteristics and then try to understand their impact on the selected performance criteria. Various researchers have studied the characteristics of malware. Nazario [11] proposes six classes of characteristics to describe computer worms: reconnaissance capabilities, specific attack capabilities, command interface, communication capabilities, intelligence and unused attack capabilities. Weaver *et al.* [18] proposes the following taxonomy of characteristics for worms: target discovery, carrier, activation, payload and attackers. Skoudis [15] compares malware to a missile and separates its characteristics in five categories: warhead, propagation engine, target selection algorithm, scanning engine, and payload.

Here, we put forth a new framework to analyse the characteristics of malware that is more general and contains all of the previous taxonomies. Furthermore, it applies to multiple domains including military and business processes. This model is also oriented towards evolution and optimisation of malware, which fits our purpose.

As before, we assume that malware attacks are conducted to reach some specific objective. We can use notions from military doctrine to describe the organisation of these at-

tacks. Like military operations, malware coordination needs command and control. The malware attacker sends *commands* to its malware agents in order to reach his objectives. The malware is *controlled* through these commands and may send feedback on its efforts using various communication channels. Many conceptual models have been developed to analyse the performance of command and control processes that describe and evaluate the capacity of an organisation to accomplish its objectives. We now describe one of the most popular and commonly used one.

4.1 The OODA loop paradigm

The concept of the *OODA Loop* was developed by Colonel John Boyd [1], in the aftermath of the Vietnam war. This American fighter pilot and military strategist described the process by which an entity reacts to an event. According to Boyd, the key to victory is to speed up the command and control process, modelled around this loop, in order to progress faster and more effectively than your opponent. The four key elements of the OODA loop (Figure 1) are, in that order: Observation, Orientation, Decision and Action.

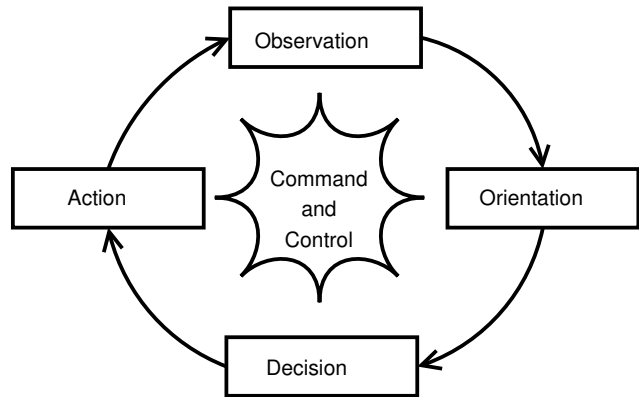


Figure 1: The OODA loop

The *observation* phase of the OODA loop answers the question “What do you see?”. It is the phase where an observer collects raw data on what is around him and what has changed since the last observation. The second phase is the *orientation*. It is the phase where the agent adds information about itself, its allies, and its past to the observation done in the first phase. The information is used to build a cognitive model of the situation in which the agent evolves. A *decision* is made based on the elements of this cognitive model and the objectives that were given to the agent prior to its mission or through commands received during. An *action* can then be taken to implement the decision made. For example, an agent could decide, based on its perception of the environment, that it is better not to attack any targets, and hence take evasive actions such as hiding its presence.

One of the advantages of Boyd’s model is the fact that it can be used to describe behaviour of agents at different levels of a process. The action of the “higher” agent is enhanced by the capacity to give orders to “lower” agents it controls. The lower agent then tries to execute requested operations and sends feedback to its controller. Figure 2 shows how OODA loops can be integrated one into another to produce a richer model. Commands by the higher level condition actions at the lower level, while observations at the lower

Fraud	Information theft	Access sale	Fame	Destruction	Information Warfare
Money	Penetration	Upstream BW	Originality	Speed	Damage
Credibility	Stealth	Security	Host location	Upstream BW	Disruption
Anonymity	Amount of information	Anonymity	Coverage	Host location	Speed
Coverage	Host location	Number	Damage	Damage	Host location
Persistence	Coverage			Coverage	Disinformation
	Persistence			Anonymity	

Table 1: Aim-oriented performance criteria

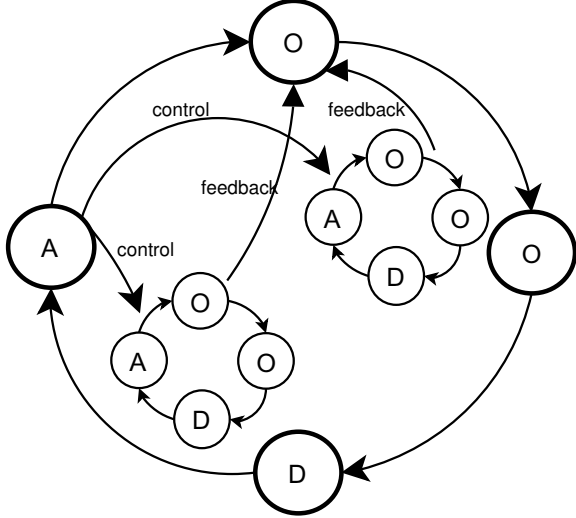


Figure 2: Nested OODA loops corresponding to different levels of command and control.

level influence observation, and hence orientation, at the higher level. In the case of malware attacks, the outer loop corresponds to the human malware attacker, while the inner loops corresponds to the deployed malware agents. This model of nested behaviours illustrates the difference between strategies of commanding and controlling agents and the tactics followed by individual agents or self-controlled groups of agents. In the case of malware attacks, the malicious attacker builds a strategy and sends order to malware instances he controls in order to achieve his aims. Meanwhile, each individual malware instance makes tactical decisions based on its own observations and cognitive model (if any), in order to best accomplish the mission and commands given by the attacker, either prior to or during the attack.

Of course, the OODA loop can also be used to describe the behaviour of defensive agents. Agent-based intrusion detection systems (IDS) have already been developed by various researchers. Within the OODA model, the observation and orientation phase are very similar. The same decision-making algorithms (rule-based, machine learning, etc.) can be applied in both realms. What varies most significantly is the range of possible actions. Actions taken by defensive agents include the isolation of an infected system, change in firewall rules, maybe even propagation to a vulnerable system and automatically patching it (e.g. as suggested in [2]).

4.2 OODA loop in malware

The OODA loop model is a good framework within which

to organise malware characteristics. The classification of characteristics according to Boyd’s model allows us to identify areas of improvement and further development in malware attacks; areas in which optimisation techniques could be applied. To do so, we will revisit the history of malware within the perspective of the OODA model.

The first malware epidemics observed on the Internet was caused by the Morris Worm [16]. It used a variety of attack vectors to compromise its Unix victims. It was also one of the few worms that used *bona fide* zero-day attacks, meaning that it exploited security flaws that were not publicly known at that time. It exploited a buffer overflow vulnerability in the *fingerd* daemon, a debugging flaw in the *sendmail* mail server and trust relationships between hosts running *rsh*. Once a victim was infected by the worm, it sent a single UDP packet to a machine on the Berkeley network before starting to propagate. This packet is thought to have been a simple way for the author to keep track of the systems that were infected by its program. From the OODA point of view, the first thing that comes to mind are the multiple attack vectors that were used by this malware. The observation phase is relatively basic and consists of checking trust relationship between hosts to discover what are the other computer systems in the neighbourhood. This information (but not the one obtained from the UDP packets) was the only one that was used in the decision process of this independent agent; there is no “processing” of this information and hence no orientation phase *per se*. Lastly, the decision process is extremely simple: if there is a vulnerable system in reach, the malware attacks it. In summary, while the attack action range of the Morris worm is relatively rich since it included various attack vectors and a simple communication method, all other components are quite poor.

Macro viruses are composed of a sequence of instructions that is interpreted, rather than executed directly. This kind of virus is interesting because it shows that passive data files can also be used as propagation mechanism for malicious software. One of the most well known epidemics of macro viruses was caused by the Melissa worm [15]. The Melissa worm infected Microsoft Word files and spread by reading the victim’s address book and e-mailing copies of itself to all the entries in the address book. When the malicious macro was executed, it checked a registry key to see if the system had already been compromised. If so, the virus would stop and otherwise it would infect the `normal.dot` template file to ensure that every document opened afterwards would contain a copy of the malicious code.

In 2003, the Slammer worm appeared [9]. This very fast worm was able to infect 90% of the vulnerable population in less than 10 minutes, by exploiting a security flaw in Microsoft SQL server. The fact that only one UDP packet had to be sent in order to exploit the vulnerability and gain

administrative privileges on the victim, greatly improved the propagation rate of this software. It generated random numbers, converted them to IP addresses, and finally attacked them using the UDP packet. No protective mechanisms were used and no active payloads were activated on the victim. The decision model of the Slammer Worm is very simple: it attacks all machines at the randomly generated IP addresses. The worm instances did not perform any orientation in order to try locate previously infected victims. This situation caused numerous repeat hits, link saturation and slowed down the worm propagation. In fact, there is no observation phase in Slammer worm at all. The worm simply guesses addresses for new targets and infects them. This characteristic obviously helped the worm to take rapid actions, but it also prevented it from achieving any other objectives that its creator might have had.

The PhatBot appeared at the end of 2003 and is a good example of modern malware. This software is coded in C++ and targets Windows systems. It uses a modular architecture to make its update as easy as possible. The PhatBot can be used for different purposes. It really is an intrusion agent that can be used for stealing information, conducting DDoS attacks or even sending spam. It uses different techniques to spread. It is possible for its controller to issue commands to make it spread to other nearby hosts or alternatively to put every malware instance in “automatic” mode where they automatically and independently find vulnerable hosts and infect them, like a network worm. Finally, the malware attacker can communicate with infected hosts using peer-to-peer communication or IRC chat rooms. From the OODA loop point of view, this malware has a very wide range of actions and observation techniques. Its communication capabilities can be placed in the decision section of the loop. Once again, the orientation section is left empty.

In addition to the characteristics displayed in these examples, one should also consider other mature techniques that may have not been so readily observed in the wild. These include passive detection avoidance techniques, like malware code polymorphism (action), the use of covert channels (action), the detection of virtual environments and defensive counter-measures (observation). In summary, Table 2 shows malware characteristics that have been observed in the wild and their classification according to the OODA loop model. Most worthy of note is the relative absence of sophistication in the orientation and decision phases of the OODA loop.

5. OPTIMISING MALWARE ATTACKS

In this section we discuss how malware could be made better. The first question to ask is why, in the absence of malicious intent, would we want to answer that question? A fair question with a simple answer: only by anticipating what future threats might be, can we hope to construct in time the protection mechanisms that will counter these threats. At the very least, one must theorise about what these threats might look like and what techniques might be used to construct them and even to optimise them.

5.1 Future trends in malware design

Indeed, we are not the first to consider this question. Staniford *et al.* [17] have described various new kinds of malware that could be very damaging to computer infrastructures. They predict the appearance of hit-list worms that carry a list of potentially vulnerable hosts and attacks them

directly, thus reducing the target identification phase and increasing the infection rate. They also describe permutation scanning worms that coordinate with other infected nodes to improve the target identification phase. The third kind of malware that they describe is the “flash worm” that uses a full list of vulnerable hosts and that could possibly infect the whole vulnerable population in a few seconds.

Nazario *et al.* [12] identify two categories of evolution for malware: improved protection and organisation between nodes. The former includes the usage of techniques such as rootkits or covert channels. As for the latter, the authors describe different organisational groupings (guerilla-like, directed tree) of infected systems to increase their coordination and reduce their chance of being discovered and stopped by network administrators. Zou *et al.* [20] describe new kinds of malware that use routing table information to only scan the Internet routable address space. They also speculate on the appearance of a worm that uses geographical information in the BGP routing tables of the Internet in order to conduct targeted attacks on a country or region.

Essentially, any such proposed improvements on malware can be divided into two broad categories, corresponding to the traditional division between strategic and tactical levels in military doctrine. The first category consists in designing and employing better attack strategies. By this, we mean that the OODA cycle at the higher levels of planning and execution of the attack utilise more sophisticated or more adequate strategies for the goals defined. In addition, the strategy chosen might be better adapted to the particular environment in which the attack will be conducted. This category is closely related to Information Warfare doctrine (or at least its more technological aspects) and probably most stands to benefit from the study of military doctrine. For example, hit-list, permutation scanning and flash worms do fall in this category of improvement and could be said to be a malware equivalent of the modern doctrine of “surgical warfare” used in the Gulf wars. In addition, efforts in this direction would also include the implementation of more sophisticated (or better adapted) command and control structures, such as those suggested in [12].

The second direction consists in improvements at the *tactical* levels of the execution of the attack, i.e. those more closely related with the deployed (and possibly autonomous) malware agents. This would include design improvements in the malware agents themselves, as well as improvements to the tools used to support the command and control functions, such as communications and information gathering and analysis. The use of increasingly sophisticated detection avoidance techniques, as suggested in [12], and the addition of observables such as routing tables, as suggested in [20], would fall into that category. In the context of malware, it is expected that improvements in this area will be technical in nature and most probably benefit by borrowing ideas from design optimisation, artificial intelligence, etc. This is the direction in which we will concentrate most of our analysis.

5.2 The malware design space

An alternative way to categorise possible future improvements to malware, in either its strategic or tactical components, is by using again the OODA model. At the strategic level, this is not a novelty since this is precisely the purpose this model was originally introduced for. At the tactical level, every behaviour of malware programmes can be con-

Observation	Orientation	Decision	Action
Trust relationships (Random) IP scanning VM detection AV/firewall detection		IRC comms. P2P comms.	Multiple exploits Multiple payloads Detection avoidance

Table 2: OODA loop and observed characteristics of in-the-wild malware

sidered a characteristic in one of the four phases of OODA.

It is useful to consider the notion of a *design space* to represent the set of malware characteristics that programmers can choose from in their quest for the most performing malicious agents. Each behavioural characteristic that a particular type of malware might show can be viewed as a particular “design subspace”. Each dimension within this subspace corresponds to a design parameter. In that way, the overall choices of active characteristics and parameter values made by the programmer in a particular malware implementation can be viewed as a point in that design space. For example, a worm might incorporate network reconnaissance behaviour: the design subspace associated with this trait would include “dimensions” such as coverage, method of reconnaissance, speed of scanning, etc. Within this metaphor, the OODA model simply allows us to organise the various characteristics (subspaces) into the four basic categories. As discussed in Section 4, many dimensions of this design space are still relatively unexplored. It is important to study in detail these areas because they represent weakness in malware design that might be improved upon in the near future.

By looking at Table 2, we see that the observation phase of malware is well developed. We have already observed various kind of techniques used to discover new targets. However, other techniques that are already used by hackers (at the strategic level), or even by defensive systems and actors, can be adopted by malware agents to improve their reconnaissance capabilities. For example, a malware agent could listen to network traffic to identify hosts on the network and the services they use. It could also be possible for a malware agent to read system logs in order to find new targets that have previously communicated with the infected system.

The action section of Table 2 also exposes that this category of behaviour is well developed in modern malware. Current malware uses various exploitation techniques to compromise new systems. They are also able to look for important information on infected systems (but not for orientation purposes). Some malware have been given the capacity to conduct DDoS attacks. On the protective side, polymorphism and process hiding techniques have been observed in an effort to avoid detection (automatic or manual) and to improve their persistence on infected systems.

On the other hand, it is in the orientation and decision phases that current malware shows most likelihood of improvement. No malware has been observed for which one could say that there was a clear process of constructing and maintaining a cognitive model of the environment and current situation. While some of the future predictions described above do imply the use of a certain representation of reality (non-routable vs. routable addresses, vulnerable vs. non-vulnerable hosts, etc.), this has not been described as a dynamic process at the tactical (agent) level. The decision logic observed in current malware is also very simplistic. All malware that has been observed to date takes direct action

based on what it has observed. At best, decisions are based on rules whose inputs are direct observations; there are no intermediate-level rules applied to processed/analysed data. None consider, for example, their past history or their potential “allies” (e.g. sites compromised by the same attack), nor do they contemplate options like waiting for a better opening or trying to find a better way to infect a network.

In both these aspects, the effectiveness of malware could be greatly improved if malware agents could carry with them a cognitive model of the network they are attacking. Beyond lists of vulnerable hosts (as suggested in [17]), consider cognitive models containing processed information about hosts that are responding, services that are available, OS and application versions, presence and location of defensive systems, network routes, or usage habits of users.

The information carried in such cognitive models could then be used to make better decision. Decisions could then be made by using any of the several decision-making and optimisation techniques that have been developed in the field of Artificial Intelligence. Already, the use of a richer cognitive model as input for rule-based reasoning would probably allow for great improvement, specially in detection avoidance and propagation speed. For example, such rules could tell a malware agent that if it finds a defensive equipment on the subnetwork he is on, he must disable it before continuing its propagation within or out of that network. The natural next step would involve the use of more sophisticated techniques from machine learning, that could be used to make malware “more intelligent” about its decisions, or more precisely allow malware agents to make better decisions about the environment they were “trained” for. In fact, many of the same techniques that have been proposed for next-generation IDS could be used in malware design. For example, fuzzy logic rules, Bayesian or neural networks could be used to fine tune the decision phase. Similarly, categorisation techniques such as data clustering could be used for profiling user or host types from data directly observable by malware agents, a kind of automated analysis from which richer cognitive models could be constructed.

5.3 Design Optimisation

The main reason for introducing the design space metaphor is in fact to allow us to illustrate and discuss the notion of malware design optimisation. In principle, each possible malware implementation is represented by a point in this space. The “goodness” of such solutions can be represented in terms of one or several of the performance criteria described in Section 3.

As we have seen, many authors expect that the most significant improvements in malware technology will come from the discovery and implementation of new behavioural traits. Such innovations correspond to an enlargement of the known design space, where the new design choices that the inclusion of such traits bring are represented by new subspaces, that

are different and in principle “orthogonal” to the subspaces of previous traits. Each new parameter of these subspaces corresponds to a new dimension in the overall space, with an associated set of possible values. From a performance standpoint, if an optimal solution has already been found within the previously known design space, the addition of new dimensions to explore might result in the discovery of a new solution of better quality in the expanded design space.

However, this point of view misses the potential improvements of applying optimisation techniques within the *known* design space. A malware creator needs to choose which characteristics to implement in his programmes and further choose the values for each possible associated parameter. As we have seen in section 4, the design space for malware is already very wide and complex, and thus it is not surprising, that a non-directed, hand-driven programming approach to malware construction has yielded such sub-optimal in-the-wild malware as we have observed. Consequently, we hypothesise that it is very likely that the use of standard optimisation techniques would yield significantly more virulent and dangerous malware, even if we restrict the design space to characteristics previously observed in the wild.

Referring back to the basic performance model of Section 2, the generic optimisation problem of malware design consists in finding the set of malware characteristics and parameter values (points in the design space), that will maximise the goodness of malware implementations, with respect to a specific performance criterion (or criteria) and within a fixed operating environment (a fixed set of values of environmental parameters). In simplified terms (i.e. Equation 1), we are fixing a performance criterion P and an operating environment E , and seeking which of set C_1 or C_2 of characteristics and parameter values maximises P . Let us consider as an example the situation where a malware programmer wants to develop malware in order to steal credit card numbers. The performance criteria could be the number of infected hosts and the amount of information gathered. The operating environment would be home machines on the Internet. Having identified the right criteria and with knowledge of the environment, he now has to decide what characteristics and values to give its malware in order to maximise them.

There are many different optimisation techniques that could be used to implement such a process. A key factor in determining their relative suitability is the ability to obtain good estimates of goodness, i.e. evaluating the chosen performance criteria. Directed approaches (gradient descent, simulated annealing, etc.) would be best suited for situations where the performance can be directly measured. Such approaches would be suitable in a situation where the characteristics of the target operating environment are well known. The performance can then be evaluated in a controlled laboratory setting where the target operating environment could be accurately emulated. In practice, such a laboratory environment could be constructed by determined malware attackers with access to relatively moderate resources (a few million dollars would suffice), by combining clusters and virtualisation technology to emulate realistic operating environment of several tens of thousands of hosts. This approach is particularly effective for the purposes of finding new vulnerabilities in known code and optimising exploits and other attack vectors for it. In fact, in this area the malware programmer could benefit from some of the

automated vulnerability detection and exploit construction methods that have been developed using advanced optimisation techniques such as genetic algorithms [4, 7]. Another higher level approach is the use of coarse-grained simulation models, which can be used to numerically predict performances of malware. This approach has been used, for example, to study the propagation characteristics of fast-spreading worms.

However, the most easily available and probably most accurate laboratory for conducting optimisation of malware is the Internet itself. First of all, there is no shortage of infectable machines that could be unwillingly recruited in this endeavour. For example, to study certain post-penetration aspects of performance, botnets of the sizes and variety observed today (several thousands of machines) [8], would probably provide a good, statistically significant representation of targets on the Internet at large. Alternatively, and for this same reason, “low-noise” test worms with propagation limited to a similar number of machines, having a lightweight, low-profile, and detection-avoiding “telemetry” payload, could provide enough feedback to drive the optimisation process. Today’s Internet is already being attacked by a variety of malware and hackers that generate high volumes of malicious traffic. Given this fact, it would be relatively easy for the malware creators to hide the whole optimisation process. In this case, the use of undirected optimisation techniques, such as those associated with machine learning, would be most effective, since only a few high-level boolean performance indicators would be observable and it would not be possible to evaluate an “accurate” goodness function based on several potentially non-observable criteria. For example, while it might be possible to see if one’s test worm showed up on the anti-virus and IDS rule updates, it might not be possible to observe where, on how many, and on what type of hosts and networks it was detected.

It is important to note that the resulting malware could be very lightweight and furthermore could be built with no obvious signs of the sophistication of the optimisation process. For example, consider the case where the cognitive model was based on a neural network of several thousand nodes, taking as input only a dozen or so easily observable inputs. Despite the fact that the training process would require large training and testing data sets, and even though the neural network itself would be large, it can be represented as a relatively simple linear function that can be easily implemented and hidden in modest amounts of binary code.

Finally, while conducting optimisation on such a large design space might be very time consuming and require considerable patience and development effort, it would require only moderate amounts of computing power. Furthermore, the optimisation techniques necessary are well-known, publicly available and now part of the curriculum of most undergraduate computer science and engineering programmes. We thus believe that it would be relatively cheap and viable for current malicious attackers to successfully mount such an optimisation process. It is very likely that this will happen in the near future, if it is not happening already...

6. CONCLUSIONS

In this paper, we discussed how malware could be optimised, and thus how their performance could be increased. In order to address that question, we have introduced a performance evaluation framework within which we can prop-

erly define what it means for malware to be “good” or “better”. We have discussed what performance criteria must be considered based on the assumption that malware or at least malware attacks have a fixed purpose. Our analysis is limited by the fact that we have considered the aims of current malware-based threats. The most dangerous threat is the one whose purpose we ignore, or alternatively one that has no rational purpose (or one we cannot understand).

We have used the OODA loop model to better organise and characterise the design criteria of malware and malware attacks. This has allowed us to identify areas in which current malware show little sophistication and in which improvements are foreseeable, we believe even in the near future. The most obvious improvements are straightforward and simple application of military doctrinal principles to the conduction of malware attacks, i.e. improvements in malware use strategy. On the other hand, we have shown that malware attacks could be improved at the tactical level by introducing in malware code better decision-making processes based on richer cognitive models of reality.

How will malware attack planners and malware developers ultimately know how to fine tune the best possible attack strategy and associated malware code? We advance that this process might also be the result of optimisation, where one or several characteristics are optimised within certain types of environment to increase performance. As such, this process might make use of well-known and time-tested optimisation techniques used in Artificial Intelligence, Data Mining or Machine Learning. Furthermore, we have argued the viability of such a process, and that it could be possibly implemented with limited resources and with little chance of detection using today’s Internet as a laboratory. An obvious next step in our research work is to implement some of these techniques and test their viability in a controlled laboratory environment. Only then will we know how really likely the appearance in the wild of such “optimised” malware will be. Furthermore, we need to elaborate on the description of the environment where malware evolve; such a description will enrich our performance analysis framework.

While we have concentrated our discussion on the optimisation of malware design, it is conceivable that optimisation techniques might also be used at the strategic level. In fact, joint optimisation of malware code design and attack strategies would yield more effective attacks. However, it is unclear to us at this moment, how the corresponding optimisation process could be easily implemented on the existing Internet infrastructure without attracting unduly attention.

However, the real threat is the one we do not know about or cannot even imagine. We are not talking about zero-day attacks based on unknown vulnerabilities (that, we can imagine...). We are talking about attacks based on completely new paradigms, be they strategic (attack planning and execution), tactical (new actions, technical innovation and coding paradigms) or both. To address this much more complex threat, we believe it is necessary to theorise and experiment with controlled co-evolution of offensive and defensive strategies and software, under varying environmental conditions. Only by subjecting both malicious and defensive software and strategies to evolutionary pressure under changes in both adversarial behaviour and environmental conditions, can we expect to observe *emergent malicious behaviour* in malware, the Unholy Grail against which we can only defend once we have observed its form.

7. REFERENCES

- [1] J. Boyd. A discourse on winning and losing. Unpublished briefing slides, 1987.
- [2] F. Castañeda, E. C. Sezer, and J. Xu. WORM vs. WORM: Preliminary study of an active counter-attack mechanism. In *Proc. ACM Work. on Rapid Malcode (WORM)*, pages 83–93, 2004.
- [3] Z. Chen, L. Gao, and K. Kwiat. Modeling the spread of active worms. In *Proc. IEEE INFOCOM*, pages 1890–1900, 2003.
- [4] C. Del Grosso, G. Antonioli, M. Di Penta, P. Galinier, and E. Merlo. Improving network applications security: a new heuristic to generate stress testing data. In *Proc. Conf. on Genetic and Evolutionary Computation (GECCO)*, pages 1037–1043, 2005.
- [5] M. Garetto, W. Gong, and D. Towsley. Modeling malware spreading dynamics. In *Proc. IEEE INFOCOM*, pages 1869–1879, 2003.
- [6] T. Holz. A short visit to the bot zoo. *IEEE Security and Privacy*, 3(3):76–79, 2005.
- [7] H. G. Kayacık, A. N. Zincir-Heywood, and M. Heywood. Evolving successful stack overflow attacks for vulnerability testing. In *Proc. of the Annual Computer Security Applications Conf. (ACSAC)*, pages 225–234, 2005.
- [8] B. McCarty. Botnets: big and bigger. *IEEE Security and Privacy*, 1(4):87–90, 2003.
- [9] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the Slammer worm. *IEEE Security and Privacy*, 1(4):33–39, 2003.
- [10] C. Nachenberg. Computer virus-antivirus coevolution. *Commun. ACM*, 40(1):46–51, 1997.
- [11] J. Nazario. *Defense and Detection Strategies against Internet Worms*. Artech House, Inc., 2003.
- [12] J. Nazario, J. Anderson, R. Walsh, and C. Connelly. The future of Internet worms. Technical report, Crimelabs Research, July 2001.
- [13] E. Rescorla. Security holes. . . who cares? In *Proc. USENIX Security Symposium*, pages 75–90, 2003.
- [14] S. E. Schechter and M. D. Smith. Access for sale: a new class of worm. In *Proc. ACM Work. on Rapid Malcode (WORM)*, pages 19–23, 2003.
- [15] E. Skoudis and L. Zeltser. *Malware: Fighting Malicious Code*. Prentice Hall PTR, 2003.
- [16] E. H. Spafford. The Internet worm program: An analysis. *Computer Commun. Rev.*, 19(1), Jan. 1989.
- [17] S. Staniford, V. Paxson, and N. Weaver. How to own the internet in your spare time. In *Proc. USENIX Security Symposium*, pages 149–167, 2002.
- [18] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham. A taxonomy of computer worms. In *Proc. ACM Work. on Rapid Malcode (WORM)*, pages 11–18, 2003.
- [19] C. C. Zou, W. Gong, and D. Towsley. Code red worm propagation modeling and analysis. In *Proc. ACM Conf. on Computer and Commun. Security (CCS)*, pages 138–147, 2002.
- [20] C. C. Zou, D. Towsley, W. Gong, and S. Cai. Routing worm: A fast, selective attack worm based on IP address information. In *Proc. IEEE Work. on Princ. Adv. and Dist. Simul. (PADS)*, pages 199–206, 2005.