Optimising Malware

[Extended Abstract]

José M. Fernandez
École Polytechnique de Montréal
P.O. Box 6079, Station Centre-Ville
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, having been used, in its various forms, with high level of success for a myriad of nefarious purposes. 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, offering an increased number of services through increasingly complex and vulnerability-ridden software.

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 actors to quickly converge towards optimal malware attack strategies and tools fine-tuned for the current Internet.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and Protection*; 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*

Malware'06 April 10-12, 2006, Phoenix, Arizona USA

General Terms

Design, Performance, Security, Theory

Keywords

Malicious software, malware attacks, malware performance, malware optimisation, OODA loop.

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

1. INTRODUCTION AND BACKGROUND

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. This generic term includes, among others, viruses, trojan horses, back doors, spyware and worms. From the onset of the widespread use of computing, malware has always represented a considerable nuisance. However, in recent years, whether as independent agents (e.g. viruses and worms) or as tools in coordinated attacks (e.g. trojan horses, backdoors, spyware), the use of malware has become one of the most insidious threats to computer security. One of the important reasons why this is so is simply because the stakes are higher. Computers and computer networks are now involved in many critical aspects of our society, from finance, management of critical infrastructure, governance, communications, biomedical applications, etc. Our wealth, our productivity, and in certain cases even our lives, depend on the services provided by these computers. As a consequence the motivation of those having villainous objectives to attack them has increased. Due to the shear size of the networks involved and the technical complexity of the software and hardware configurations involved, the use of malware tools, whose generation and use has become automated to various degrees, has become a prevalent method among the many tools used to attack such systems.

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. To address this question, let us consider briefly some of the most significant events of the past history of malware.

Malware has been present in computers for a very long time. The first virus found in the wild targeted the Apple II computer around 1981. In November 1988, the early Internet was paralysed by the first worm epidemic caused by the Morris worm [15], named after its author's name, Robert

Tappan Morris Jr. Besides being the first worm, it is important to note that the Morris worm was quite sophisticated, incorporating several different attack vectors. Nonetheless, detailed analysis of its code showed that it contained several flaws and was relatively inefficient. Two years later, the first in-the-wild viruses using code polymorphism and other anti-virus evasion techniques saw the day, indicating a significant evolutionary step: a primitive sense of awareness and ability to adapt to the environment. Ten years later, a group called the "Cult of the Dead Cow" released its famous backdoor programme, Back Orifice, allowing a malicious actor to remotely control a Windows system without the knowledge of its victim. The introduction of such tools (of which Back Orifice was only to be the first of a long list) allowed malicious actors to re-enter the attack process in the middle of its execution, by giving them the ability to remotely control infected machines and thus (theoretically) adapt the execution of an attack according to changes observed in the environment.

In 2003, the fastest ever propagation of malware was observed with the Slammer worm [8]. This program infected 90% of the vulnerable population in less than 10 minutes using a single UDP packet as its attack vector. Curiously, things could have been worse: a programming flaw in the pseudo-random number generator in Slammer made it inadvertently repeat-hit many previously infected targets. Finally, the last two years have seen an increasing use of *botnets* [7], networks of infected machines controlled and managed by automated tools, that allow its "0wners" to use them for their purposes, such as sending unsolicited e-mails and conducting distributed denial-of-service (DDoS) attacks.

What can be learned from History? For one, that writing good and completely autonomous malware (specially worms) is probably very hard... This is probably one of the reasons why the most common (and probably the more 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 [5]. Second, that most significant developments have been concentrated on the penetration, propagation and detection avoidance aspects.

As far as penetration is concerned, the large numbers of available attack vectors [12] 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 and/or unpatched vulnerabilities and misconfiguration errors to be exploited. On the other hand, 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 proposed 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 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 [9] between malware and malware counter-measures (honeypots and/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. 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, a well-known paradigm of military doctrine used to describe command and control functions, to organise and classify the various aspects of malware attacks strategy and malware design that constitute the decision and design space of malware construction and use. The use of the OODA-loop model allows us to 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 actor with the help of malicious software in order to achieve a precise purpose. In such attacks, the malicious actor 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 actor or software he is using[1]. 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 of 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 rate, infection coverage, bandwidth utilised, etc. Which criteria are pertinent depend on the objectives of the malware actor. For example, if a malicious actor conducts a malware attack to steal credit card numbers, the performance of this attack will be evaluated on the number of stolen credit card numbers and the preservation of the anonymity of its author. If as part of that attack he writes a Trojan horse that is sent to potential victims to obtain their credit card numbers, the performance of that particular malware might be measured in terms of attraction rate. Finally, if a malicious actor wants to construct a botnet to perform DDoS attacks, the performance of the attack will be evaluated according to the number of hosts that have been 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 criterion that the attack might be evaluated by. More examples and a general description of the relevant malware characteristics is given in 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 him.

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 actor 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. This main thesis is the basis of our model and is illustrated in Figure 1. Thus, typical statements about malware performance could take one of the forms below:

[1]Note that in our definition, for example, a software tool that helps a malicious actor control and manage a botnet is not itself malware, even though the use of the botnet constitutes a malware attack because the presence of malicious software on the infected machines (backdoors). Similarly, in the absence of malware slave clients, or if this client software is non-malicious (e.g. on a computing grid), then there is no malware attack *per se*.

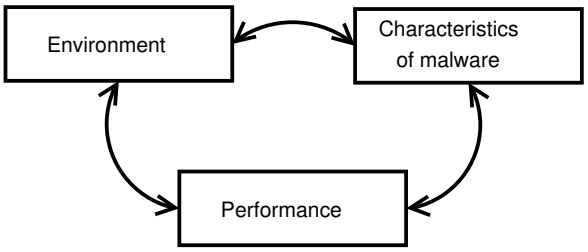

Figure 1: Basic performance model of malware

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

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

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

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

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 actors 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 hence a different associated performance criterion P_2), as the strategy and tactics of the attack (represented by C) vary.

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. AIM-ORIENTED PERFORMANCE ANALYSIS

In order to evaluate the performance of malware attacks, we need to establish performance criteria, as described in the previous section. We base our analysis of the performance of malware on the assumption that all malware attacks have a purpose. By that we mean that an attack is planned and performed to achieve a certain, definite goal or goals. The performance criteria become the metrics of achievement of the aim behind the malware attack.

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.*[16] have used the "classical epidemic model"to analyse the spread of the Code Red worm. Zou *et al.*[18] 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.*[4] have presented an analytical technique based on Markov Chains to capture the spreading characteristics of malware.

However, our objective is to introduce a more general theory of malware performance that considers other plausible performance criteria. Here we are more interested in being more thorough in the qualitative identification of all (or most) pertinent criteria, albeit at the detriment of quantitative results that may be obtained from some of the mathematical models in previous work (possibly in the forms of Equations 1–4). We are not saying that such models and quantitative precision are not desirable and worthy of study, but rather that (as we will argue in Section 5) from the malware designer's point of view, minimal experimental data in the same form might be sufficient and an adequate alternative for malware optimisation purposes.

As stated, we base our analysis on the fact that performance is a metric 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 five main motivations to write malware.

1. Fraud,
2. Information theft,
3. Sale of access to computing resources,
4. Plain destruction (cyber vandalism), and
5. 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. We have thence divided the performance criteria we have found into two broad categories: *generic performance criteria*, that are applicable to all of the above objectives, and *specific performance criteria*, that are applicable to one, or a few, of the above motivations.

3.2 Generic performance criteria

Even if malware can be used to fulfil various objectives, we can extract three performance criteria that can always by

observed on malware and that are useful to evaluate performance: *number of infected hosts*, *persistence*, and *anonymity*.

The **number of infected hosts** represents the sum of systems that were infected by the malware under evaluation. This performance criteria will influence the bandwidth available for conducting DDoS attacks, number of information that can be stolen from a network, communication infrastructures, etc. One of the basic characteristic of malware is the fact that it spreads from computer to computer automatically. The number of infected hosts metric shows how effective the target discovery and propagation methods are. A high number of infected hosts will denote a malware that has good propagation mechanism but could decrease its stealthiness. Note that while this is essentially a strategic metric, not only can it be affected by strategy decisions by malware attacker but also by tactical actions taken by the malware itself.

Persistence of a malware is its ability to stay present on an infected system. It is a tactical performance metric associated with the actual malware. In other words, malware is persistent if it is hard to disinfect a system that it has compromised. It also value at the strategic level, because the persistence of malware is profitable to its creator as it guarantees that his piece of software keeps working for him. Various techniques can be used to ensure persistence on a system. Process hiding or kernel modification are examples of malware characteristics that make malware hard to detect and destroy.

Anonymity of a malware creator or controller means that his true identity can not be discovered. Anonymity is important for malicious users to avoid being brought to justice or avoid revenge from victims. To ensure anonymity, a malware programmer needs to leave no traces of its identity in its code. Furthermore, he needs to be very cautious when he communicates with his malware instances in order to be hard to trace. Hence, this performance criterion is influenced by both tactical and strategic level actions.

3.3 Aim-specific performance criteria

We know turn to describe the performance criteria specific to each of the five objective categories we have identified.

Fraud. Cybercriminals are using malware to conduct fraud, there is no question about. This aim is clearly getting more and more popular amongst malware programmers. Fraud is a deception for personal gain, and it aims toward obtaining money or property by false pretences. 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 the targeted user; the associated performance criterion is the malware **credibility** or **deception ratio**.

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

Access for Sale. 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 [13]. Such botnets have been used in the past to send unsolicited e-mails, conduct distributed denial of service attacks, and perform large scale computation. The performance of malware attacks aimed for access sale is impacted by the **total available upstream bandwidth** (specially for conducting DDoS attacks) and the **security of access** to the infected hosts. The available bandwidth can be calculated by summing the bandwidth available to each infected host. The access security 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.

Destruction. In terms of numbers, most malware attacks on the Internet have been aimed toward destruction, whether through targeted (DDoS attacks, web site defacings) or non-targeted (paralysing viruses and worms). In addition, even certain malware observed in the wild without any active payload, have caused by their mere propagation traffic serious damage to network infrastructures [8]. A higher number of infected hosts will allow the attacker to perform wider attacks. Also, the **speed of propagation** or **propagation rate**, will impact how much damage an epidemics can do. The **total upstream bandwidth** of the infected network will increase DDoS-attack capability and the **location of infected hosts** will also change the impact of the destruction. 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. In all cases, another specific performance criterion would the total amount of **hardware and material damage** inflicted as a direct or indirect consequence of the malware attack.

Information Warfare. Information Warfare is the extension of the battlefield to information technology. In doctrinal terms, it is the offensive and defensive use of infor-

mation 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. For Information Warfare, performance criteria include the **speed of action**, the **location of infected hosts** and amount of disturbance or **hardware and material damage** inflicted. The speed of action is important to lower the chances of reaction by the defender. The disturbance is the action to disrupt information systems or to use them for malicious purposes; these effects need to be measured. Finally, the location of infected hosts will impact the amount of damage inflicted, as discussed above, but also on 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** (i.e. number of client stations that will surf them) of targeted web sites that have been defaced or altered.

In summary, we have identified here 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. Again, 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 attack really knows...

4. CHARACTERISATION OF MALWARE ATTACKS

In order to perform performance analysis on malicious software, we need to establish characteristics that will be observed and try to understand their impact on performance criteria. Various researchers have studied the characteristics of malware. Nazario [10] proposed 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* [17] proposed a different taxonomy. They use the following characteristics to analyse worms: target discovery, carrier, activation, payload and attackers. Skoudis [14] compares malware to a missile and separates its characteristics in five categories: warhead, propagation engine, target selection algorithm, scanning engine, and payload.

Even if various malware taxonomy have been developed, we bring forward a new framework to analyse the characteristics of malware. Our model is more general and can contains all of the previous taxonomies. Furthermore, the model we use to analyse malware applies to multiple domains including military and business processes. This model is also oriented towards evolution and optimisation which fits our purpose.

As stated before, we assume that in general malware, like any other category of software, are written for a purpose. This means that malware are built to fulfil an objective.

Knowing that malware are independent agents and exist to reach some objectives, we can use notions from military doctrine to describe the organisation of their attacks. Like military operations, malware coordination needs command and control. With command and control, the creator of malware sends commands to its resources in order to reach its objectives. Its units are controlled by his orders and send feedback of their effort using pre-established communication channels. Many models have been developed to analyse the performance of command and control. These models describe and evaluate the capacity of an organisation to accomplish its objectives. One of the most commonly used model in the military for command and control is the *OODA loop*.

4.1 The OODA loop paradigm

The concept of *OODA Loop* was developed by Colonel John Boyd [2] at the end of the twentieth century. This American fighter pilot and military strategist described the process by which an entity reacts to an event. According to this concept, the key to victory is to speed up the decision process in order to progress faster than your opponent.

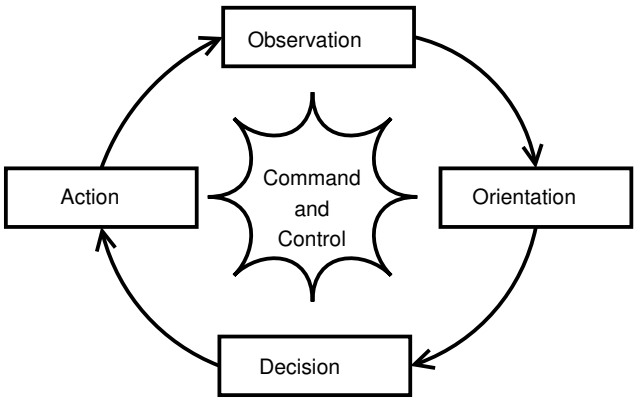

Figure 2: OODA loop

Figure 2 gives an overview of the key components of the *OODA loop*. The four key elements are observation, orientation, decision and action.

The *observation* phase of the *OODA loop* answers the question "What do you see?". It is the phase where an observer collects raw data of 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 done based on the cognitive model elaborated in the orientation phase and the objectives that were given to the agent prior to its mission. An action can then be taken to implement the decision made, which constitutes the last step of the OODA loop, the *action* phase. The action phase includes every operation an agent can perform. For example, an agent could decide, based on the knowledge of its environment that it is better to defend itself by hiding instead of attacking the enemy.

One of the advantage 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 with the capacity to give orders to other agents he controls. The "lower" agent then executes requested operation and sends feedback to its controller. Figure 3 shows how OODA loops can be integrated one into another to produce a richer

Generic	Fraud	Information theft	Access sale	Destruction	Information Warfare
Number of hosts	Money	Penetration	Upstream BW	Propagation	Speed
Persistence	Credibility	Stealth	Security	Upstream BW	Host Location
Anonymity		Amount of information		Host Location	Damage
		Host location		Damage	Exposure

Table 1: Generic and specific aim-oriented performance criteria

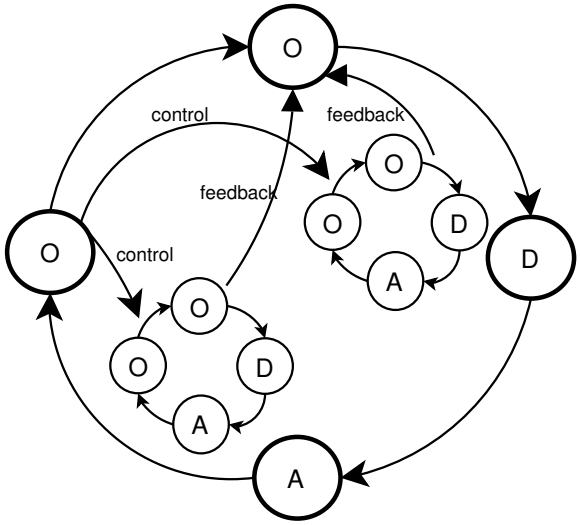

Figure 3: Nested OODA loops corresponding to different levels of command and control.

model. Commands by the higher level condition actions at the lower level, while observations at the lower level influence observation, and hence orientation, at the higher level. In the case of malware attacks, the outer loop corresponds to the human malware actors, while the inner loops corresponds to the deployed malware agents.

With the nesting of behaviour model, we can get a clear picture of the situation that includes strategies of controlling agents and tactics of operation units. In the case of malware attacks, the malicious actor build a strategy and sends order to malware instances he control to reach his objectives.

4.2 OODA loop and malware

The OODA loop model is useful to us because it is a good framework 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. After identifying areas of possible development, we can predict possible optimisations.

The OODA loop model helps us shed new light on the characteristics of malware from a generic point of view. The first step in the spread of malware is the identification of a target. When a vulnerable system is identified, the malware tries to penetrate it with its attack vector. If the host is compromised, the malware sends a copy of itself to the newly infected system and thus propagates. After replication, malware instances can elaborate defences or communicate between them. Finally, the malware activates its payload in order to achieve its objective.

The identification of a new target consists in finding other systems that could be infected with the malware. This can be done using network scanning techniques to find hosts that are active on the network or by reading the address book of a newly infected system. Looking for targets thus involves *observation* of various types of data, such as network traffic, OS fingerprints, etc. However, making a decision about what *is* a target is itself part of the *orientation* process.

The attack phase of a malware include the actions that are taken to obtain execution control on the target system. In the case of computer worm, this action is performed using a vulnerability in one of the services that is active on the victim. In the case of a virus that spreads via e-mail, the virus is going to send itself with an e-mail in hope of being executed by one of its potential victims. Attacking is clearly part of the action phase described by Boyd.

Malware propagation is the operation by which it copies itself to a newly infected system. This can be done, for example by sending an executable file via e-mail or downloading it from a web site. Some malware also copy their programme using FTP transfers from the attacking node to the newly infected victim. We include the propagation in the set of actions that can be taken by a malware.

Different methods can be used by malware to establish communication with their creator or to coordinate between infected nodes. For example, an IRC network is often used by botnet creators [7] to communicate with infected systems and issue command or upload new modules for their programs. In relation with the OODA loop, communication can be placed all phases of the OODA loop. Communicating the result of observation, analysis (orientation) or action, either to the upper level of command and control or to other malware agents, is itself an *action* than can be taken by malware. Conversely, Communication is seen as an action when it is used to issue commands to the infected hosts. In the absence of a decentralised command and control structure, which is the norm in most malware attacks up to now, communication through more or less stealthy means such as IRC or peer-to-peer networks is the only way to transmit the result of command *decisions* taken by the malware attacker. On the other hand, communication is part of the *orientation* process when information concerning other infected nodes is broadcasted in order to coordinate action. Finally, communications between infected systems and/or defensive systems can itself be the object of *observation*.

Defensive actions that can be taken by a malicious program include operations to hide from the system user or administrator, or make it harder to be deleted. Stealth on the network side is also considered as part of the defensive actions malware can use to defend themselves against network intrusion detection or prevention systems.

The last characteristic of malware is related to their payload. Payload is the programme code that is carried by the malware and that is executed on the newly infected host after the propagation phase in order to achieve the ultimate

purpose of the attack. For example, in the case of a botnet, the payload of the malware is to install an IRC client and to connect to a predefined IRC server in order to receive more information from its creator. Payload activation is also part of the action malware can perform.

Table 2 shows the different phases of a malware life and compares them to the behaviour of an agent in the *OODA loop* model.

4.3 History of malware revisited

Now that we have a rich model to illustrate complex situation of command and control, we can apply it to analyse in hindsight some of the most notorious malware epidemics that have been seen on the Internet.

The first worm epidemic that was observed on the Internet was caused by the Morris Worm [15]. This malware was released on the Internet in November 1988. The Morris worm attacked Unix systems that were connected to the Internet. It used a variety of attack vectors to compromise its victims. It was also one of the only worm that used a zero-day attack, meaning that it exploited a security flaw that was not publicly known at that time. This malicious software exploited a buffer overflow vulnerability in the *fingerd* daemon, a debugging flaw in sendmail's mail server and trust relationship 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.

If we evaluate the characteristic of the Morris worm from Boyd's point of view, the first thing that comes to mind are the multiple attack vectors that were used by this malware. We can classify this range of attacks in the action section of the *OODA loop*. The observation phase is pretty basic and consists of checking trust relationship between hosts to discover what are the other computer systems in the neighbourhood. The information gathered in the observation phase is the only one that is used in the decision process of this independent agent. Furthermore, the decision process is extremely simple: if there is a vulnerable system in reach, the malware attacks it. To sum up, the action range of the Morris worm is rich because it includes various attack vectors and a simple communication method. On the other side,all other components are very poor. The observation and decision phase are very unsophisticated.

A macro virus is a virus composed of a sequence of instructions that is interpreted, rather than executed directly [1]. 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 virus was caused by the Melissa worm[14]. 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 is infected, it checks a registry key to see if the system has already been compromised. If so, the virus stops, otherwise it infects the `normal.dot` template file to ensure that every document opened from this document is infected.

In 2003, the Slammer worm appeared [8]. This very fast worm was able to infect 90% of the vulnerable population in less than 10 minutes. This program exploited a security flaw in Microsoft's 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 speed of this software. This worm generated random numbers, converted them to IP addresses and finally attacked them using it's UDP packet. No defence mechanism were used and no active payload were activated on the victim.

The decision characteristics of the Slammer Worm are simple, it generates a random IP address with a random number generator and then attacks the machine at that address. The worm instances do not perform any orientation phase to try to see where other infected instances can be located. This absence of orientation caused link saturation and slowed down the worm propagation. Furthermore, the observation process for the Slammer Worm does not exists. The worm simply guesses addresses for new targets and infects them. This characteristic obviously helped the worm to take rapid action but also has the consequence of not reaching any other objectives than infecting as many systems as possible.

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 denial of service attack or even sending spam e-mails. This malware uses different techniques to spread. It is possible for its controller to issue commands to make it spread to other nearby hosts or 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 action and of observation techniques. Its communication capabilities can be placed in the decision section of the loop. Once again, the orientation section is left empty.

Using our general model to describe the behaviour of malware attacks, we have used the four phases of the OODA loop to classify their various characteristics of observed malware. Table 3 shows malware characteristics that have been observed in the wild and their classification according to this model. The classification of malware characteristics in an objective-oriented context shows that wide areas of the design space for malware have not yet been observed on the Internet.

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? And further, why would we want to build better malware?[2]. A fair question with a simple answer: only by anticipating what future threats might be, can we hope to construct in time the defensive mechanisms that will counter these

[2]Indeed, as the much more colourful Spanish version of the English proverb "You lie down with dogs, you wake up with flees," says, "If you raise ravens, they will eat your eyes out."

Observation	Orientation	Decision	Action
Network scan OS fingerprinting VM detection	Target identification	Communication	Attack Propagation Defence Payload activation

Table 2: OODA loop and generic malware characteristics

Observation	Orientation	Decision	Action
Trust relationship Random IP scan Local IP scan		IRC communication Peer-to-peer	Various exploits Denial of service Personal information theft Process hiding Spamming Polymorphism

Table 3: OODA loop and observed characteristics of in-the-wild malware

threats [3]. 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.* [16] 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 of its propagation. 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.* [11] identify two categories of evolution for malware: improved defences and organisation between nodes. These improved defences includes the usage of techniques such as rootkits or covert channels. As for the improvement of coordination between infected nodes, 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.* [19] describe new kinds of malware that use routing table information to only scan the Internet routable address space. The authors 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 [11].

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 [11] and the addition of additional observables such as routing tables, as suggested in [19], 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-loop model. At the strategic level, this is not a novelty since this is precisely what this model was originally introduced to describe. At the tactical level, every programmable behaviour of malware can be considered as a characteristic of malware in one of the four categories of OODA.

It is useful to consider the notion of a *design space* to represent the set of malware characteristics that malware programmers can choose from in their quest for the most performing tools and malware agents. Each behavioural characteristic that a particular type of malware might or might not show can be viewed as a particular "design subspace", with each dimension within corresponding to a relevant as-

[3] In other words, *Si vis pacem, para bellum.*

sociated design parameter. In that way, the overall choices of active characteristics and particular parameter choices made by the programmer in a particular malware implementation can be viewed as point in that space. For example, a worm might incorporate network reconnaissance behaviour: the design subspace associated with this characteristic 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 relatively unexplored by current malware. 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 3, 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 3 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. Some malware have been given the capacity to conduct DDoS attacks. On the defensive side, polymorphism and process hiding techniques are observed as an effort to avoid detection (automatic or manual) and to improve their persistence on infected system.

On the other hand, it is in the orientation and decision phases that current malware shows most potential for improvement. No observed malware has been observed for which one could say that were 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 take direct action based on what they have observed; at best it is based on rules whose input are direct observations by the system; there are no intermediate-level rules applied to processed/analysed data. None of them 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 the list of vulnerable hosts (as suggested in [16]), consider cognitive models containing processed information about the following:

- Hosts that are responding;
- Services that are available on the network;
- Operating system and application version of hosts;
- Presence and location of defensive equipment and software;
- Routes taken by network traffic; and
- Usage habits of users

The information carried in such cognitive models could then be used to take better decision. Decisions could then be taken by using any of the several decision-making and optimisation techniques that have been developed in the field Artificial Intelligence (AI) techniques that have been developed in this field of research (see for example [6]). Already, the use of simpler rule-based reasoning as input the components of a richer cognitive model, such as the above, rather than direct local observations 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 design space, where the new design choices that the inclusion of such traits bring are represented by new "orthogonal" subspaces, where each new parameter corresponds to a new dimension in the overall space and 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.

However, this point of view misses the potential improvements of applying optimisation techniques within the known design space. As we have seen in section 4, the design space for malware is very wide. A malware creator needs to choose which characteristics to implement in his programmes and

further choose the values for each possible associated parameter (including not using them). With the complexity of the design space, it is not surprising, that a non-directed, human-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.

In broad terms, and in terms of the basic model of Section 2, the generic optimisation problem of malware design consists in finding the set of malware characteristics and parameters (points in the design space), that will maximise the goodness of malware implementations with those characteristics and parameters, 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 and referring back to Equation 1, we are fixing a performance criterion P and an operating environment E, and seeking which of the set of characteristics and parameters, C_1 or C_2 maximises P.

Let us consider as an example the situation where a malicious programmer wants to develop malware that is going to infect as many systems as possible in order to steal credit card numbers stored on disk. The performance criteria will be the number of infected hosts and the amount of information gathered. The operating environment would be the Internet itself. Knowing the performance criteria and environment for its design, the malware creator now has to decide what characteristics he will give to its malware.

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 best be 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 emulated. In practice, such a laboratory environment could be constructed by determined malware actors with access to a 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. 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 to study the propagation characteristics of flash and other types 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), would probably provide a good, statistically significant representation of targets on the Internet at large. However, their usefulness would be limited to study such performance characteristics as penetration and propagation rates.

Again, because a sampling of several thousand machines would probably be statistically significant, "low-noise" test worms propagating to a limited number of machines and 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. This malicious traffic can be used by malware creators to hide the propagation of their test specimen. In fact, it would be relatively easy for the malicious actor to hide the whole optimisation process on today's Internet. 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. 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 and on what type of network 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 of a malicious agent was implemented as a neural network of several thousand nodes, taking as input a few tens of easily observable inputs. While the training process would require large training and testing data sets, and the neural network itself is quite large, the final solution is a simple linear function that can easily be implemented and hidden in machine 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 relatively cheap and viable for current malicious actors to successfully mount such an optimisation process and is very likely to 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 properly define what it means for malware to be "good" or "better". This framework takes into consideration the three following aspects: the environment where a malware attack is performed, the malware attack characteristics (both strategic and tactical), and the aim-oriented performance criteria associated with particular objectives of the attack. 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, a well-known model of representing command and control process in military doctrine, 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 conduction of malware attacks. On the other hand, we showed that malware design could be improved by introducing better decision-making 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? We advance that this process might also be the result of a 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, Machine Learning, and Operations Research. Furthermore, we have argued the viability of such a process, which, could possibly be implemented with limited resources and with little chance of detection using today's Internet as a laboratory. While here 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 principle, joint optimisation of malware design parameters and attack strategy would yield more effective attacks. However, it is unclear to us at this moment the corresponding optimisation process could be easily implemented on the existing Internet infrastructure without attracting unduly attention.

An obvious next step in our research work is to implement some of these techniques and test their viability. 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.

However, the real threat is the one we don't know about or cannot even imagine. We are not talking about zero-day attacks based on unknown vulnerabilities in known software (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 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] M. Bishop. *Computer Security: Art and Science*. Addison-Wesley, Boston, USA, 2003.

[2] J. Boyd. A discourse on winning and losing. Unpublished briefing slides, 1987.

[3] Z. Chen, L. Gao, and K. Kwiat. Modeling the spread of active worms. In *IEEE INFOCOMM*, 2003.

[4] W. T. D. Garetto, M.; Gong. Modeling malware spreading dynamics. In *INFOCOM 2003*, pages 1869 – 1879. IEEE, 2003.

[5] T. Holz. A short visit to the bot zoo [malicious bots software]. *IEEE Security and Privacy Magazine*, 3(3):76–79, 2005.

[6] G. F. Luger and W. A. Stubblefield. *Artificial intelligence (2nd ed.): structures and strategies for complex problem-solving*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1993.

[7] B. McCarty. Botnets: big and bigger. *IEEE Security and Privacy Magazine*, 1(4):87–90, 2003.

[8] 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.

[9] C. Nachenberg. Computer virus-antivirus coevolution. *Commun. ACM*, 40(1):46–51, 1997.

[10] J. Nazario. *Defense and Detection Strategies against Internet Worms*. Artech House, Inc., Norwood, MA, USA, 2003.

[11] J. Nazario, J. Anderson, R. Walsh, and C. Connelly. The future of internet worms, 2001.

[12] E. Rescorla. Security holes... who cares? In *Proceedings of the 11th USENIX Security Symposium*, pages 75–90. USENIX, Aug. 2003.

[13] S. E. Schechter and M. D. Smith. Access for sale: a new class of worm. In *WORM '03: Proceedings of the 2003 ACM workshop on Rapid Malcode*, pages 19–23, New York, NY, USA, 2003. ACM Press.

[14] E. Skoudis and L. Zeltser. *Malware: Fighting Malicious Code*. Prentice Hall series in computer networking and distributed systems. Prentice Hall PTR, 2003.

[15] E. H. Spafford. The Internet worm program: An analysis. *Computer Communication Review*, 19(1), Jan. 1989.

[16] S. Staniford, V. Paxson, and N. Weaver. How to own the internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium*, pages 149–167, Berkeley, CA, USA, 2002. USENIX Association.

[17] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham. A taxonomy of computer worms. In *WORM '03: Proceedings of the 2003 ACM workshop on Rapid Malcode*, pages 11–18, New York, NY, USA, 2003. ACM Press.

[18] C. C. Zou, W. Gong, and D. Towsley. Code red worm propagation modeling and analysis. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 138–147, New York, NY, USA, 2002. ACM Press.

[19] C. C. Zou, D. Towsley, S. Cai, and W. Gong. Routing worm: A fast, selective attack worm based on ip adress information. Technical report, 2003.