FOI-R–0840–SE
March 2003
ISSN 1650-1942

Scientific Report

Martin Karresand

A Proposed Taxonomy of Software Weapons

Command and Control Systems
P.O. Box 1165
SE - 581 11 LINKÖPING

FOI – SWEDISH DEFENCE RESEARCH AGENCY
Command and Control Systems
P.O. Box 1165
SE - 581 11 LINKÖPING

FOI-R–0840–SE
March 2003
ISSN 1650-1942

Scientific Report

Martin Karresand

A Proposed Taxonomy of Software Weapons

Acknowledgements

I would like to thank Arne Vidström for sharing his deep knowledge of software weapons with me and for always being prepared to discuss definitions, formulations, and other highly abstract things.

I would also like to thank my supervisor Mikael Wedlin and my examiner Viiveke Fåk for their support and for having confidence in me.

Likewise I would like to thank Jonas, Helena, Jojo and the rest of my class, as well as my other friends, for brightening my life by their presence.

And last but not least I would like to thank my beloved fiancée Helena for always supporting me no matter how hard I studied. I love you from the bottom of my heart, now and forever.

Issuing organisation
FOI - Swedish Defence Research Agency
Command and Control Systems
P.O. Box 1165
SE - 581 11 LINKÖPING

Report number, ISRN
FOI-R–0840–SE

Report type
Scientific Report

Month year
March 2003

Project number
E7033

Customers code
5. Commissioned Research

Research area code
4. C4ISR

Sub area code
41. C4I

Author(s)
Martin Karresand

Project manager
Mikael Wedlin

Approved by
Lennart Nyström

Sponsoring agency

Scientifically and technically responsible
Viiveke Fåk

Report title
A Proposed Taxonomy of Software Weapons

Abstract
The terms and classification schemes used in the computer security field today are not standardised. Thus the field is hard to take in, there is a risk of misunderstandings, and there is a risk the scientific work is being hampered.
Therefore this report presents a proposal for a taxonomy of software based IT weapons. After an account of the theories governing the formation of a taxonomy, and a presentation of the requisites, seven taxonomies from different parts of the computer security field are evaluated. Then the proposed new taxonomy is introduced and the inclusion of each of the 15 categories are motivated and discussed.
The final part of the report contains a discussion of the general defences against software weapons, together with a presentation of some open issues regarding the taxonomy. Finally the report is summarised.

Keywords
IT weapon, taxonomy, malware, IW, information warfare, computer security, virus, worm

Further bibliographic information

ISSN
ISSN 1650-1942

Pages
154

Language
English

Price
Acc. to price list

Security classification
Unclassified

Utgivare	Rapportnummer, ISRN	Klassificering
Totalförsvarets Forskningsinstitut - FOI Ledningssystem Box 1165 581 11 LINKÖPING	FOI-R–0840–SE	Vetenskaplig rapport
	Månad år mars 2003	Projektnummer E7033
	Verksamhetsgren 5. Uppdragsfinansierad verksamhet	
	Forskningsområde 4. Spaning och ledning	
	Delområde 41. Ledning med samband, telekom och IT-system	

Författare	Projektledare
Martin Karresand	Mikael Wedlin
	Godkänd av Lennart Nyström
	Uppdragsgivare/kundbeteckning
	Tekniskt och/eller vetenskapligt ansvarig Viiveke Fåk

Rapporttitel

Ett förslag på taxonomi för programvarubaserade IT-vapen

Sammanfattning

De termer och klassificeringsmodeller som idag används inom IT-säkerhetsområdet är inte standardiserade. Det är därför svårt att överblicka området, det finns en risk för missförstånd och det vetenskapliga arbetet riskerar att bli lidande.

För att råda bot på dessa problem har ett förslag på en taxonomi för programvarubaserade IT-vapen tagits fram, vilket presenteras i den här rapporten. Först presenteras dock de teorier som ligger till grund för skapandet av en taxonomi, samt de behov av en taxonomi som identifierats under arbetets gång. Därefter utvärderas sju taxonomier inom angränsande områden. När detta är gjort introduceras den nya taxonomin och dess 15 kategorier motiveras och förklaras.

Den sista delen av denna rapport diskuterar kortfattat de generella försvarsmetoder som finns gentemot programvarubaserade IT-vapen. Detta följs av en presentation av den framtida utvecklingen av taxonomin. Slutligen sammanfattas rapporten i en slutdiskussion.

Nyckelord

IT-vapen, taxonomi, malware, IW, informationskrigföring, IT-säkerhet, virus, mask

Övriga bibliografiska uppgifter

ISSN	Antal sidor	Språk
ISSN 1650-1942	154	Engelska
Distribution enligt missiv	Pris Enligt prislista	
	Sekretess Öppen	

Contents

List of Figures

List of Tables

1 Introduction

The computer security community of today can be compared to the American Wild West once upon a time; no real law and order and a lot of new citizens. There is a continuous stream of new members pouring into the research community and each new member brings his or her own vocabulary. In other words, there are no unified or standardised terms to use.

The research being done so far has mainly been concentrated to the technical side of the spectrum. The rate of development of new weapons is high and therefore the developers of computer security solutions are fighting an uphill battle. Consequently, their solutions tend to be pragmatic, many times more or less just mending breaches in the fictive walls surrounding the computer systems.

As it is today, there is a risk of misunderstanding between different actors in the computer security field because of a lack of structure. By not having a good view of the field and no well defined terms to use, unnecessary time might be spent on making sure everyone knows what the others are talking about.

To return to the example of the Wild West again; as the society evolved it became more and more structured. In short, it got civilised. The same needs to be done for the computer security society. As a part of that there is a need for a classification scheme of the software tools used for attacks.

Also the general security awareness of the users of the systems will benefit from a classification scheme where the technical properties of a tool are used, because then they will better understand what different types of software weapons actually can do. They will also be calmer and more in control of the situation if the system is attacked, because something known is less frightening to face, than something unknown.

One important thing is what lies behind the used terms, what properties they are based on. The definitions of malware used today all involve intent in some way, the intent of the user of the malicious software, or the intent of the creator of the software. Neither is really good, it is really impossible to correctly measure the intents of a human being. Instead the definition has to be based on the tool itself, and solely on its technical characteristics. Or as Shakespeare let Juliet so pertinently describe it in Romeo and Juliet[1] [2, ch. 2.2:43–44]:

> What's in a name? That which we call a rose
> By any other word would smell as sweet;

Therefore this proposed taxonomy of software weapons might have a function to fill, although the work of getting it accepted may be compared to trying to move a mountain, or maybe even a whole mountain range. But by moving one small rock at a time, eventually even the Himalayas can be moved, so please, continue reading!

[1] The citation is often given as '[...] any other *name* [...]', which is taken from the bad, 1st Quarto. The citation given here is taken from the good, 2nd Quarto. [1]

1.1 Background

During the summer of 2001 a report [3] presenting a proposal for a taxonomy of software weapons (software based IT weapons[2]) was written at the Swedish Defence Research Agency (FOI). This report was then further developed in a paper that was presented at the 7th Nordic Workshop on Secure IT Systems (NordSec 2002) [4].

The proposal was regarded as interesting and therefore a deepening of the research was decided upon in the form of a master's thesis. The project has been driven as a cooperation between Linköping University, Sweden, and FOI.

1.2 Purpose

The purpose of this thesis is to deepen the theoretical parts of the previous work done on the taxonomy and also empirically test it. If needed, revisions will be suggested (and thoroughly justified). To facilitate the understanding of the thesis the reader is recommended to read the NordSec paper, which is included as an appendix (see Appendix A).

Also the general countermeasures in use today against software weapons with the characteristics described in the taxonomy will be presented.

1.3 Questions to be answered

The thesis is meant to answer the following questions:

- What are the requirements connected to the creation of a taxonomy of software weapons?
- Are there any other taxonomies covering the field and if so, can they be used?
- What use do the computer security community have for a taxonomy of software weapons?
- Are the categories in the proposed taxonomy motivated by the above mentioned purpose for creating a taxonomy?
- How well does the taxonomy classify different types of weapons?

[2]The Swedish word used in the report is 'IT-vapen' (IT weapon). This term has another, broader meaning in English. Instead the term *malware* (malicious software) is used when referring to viruses, worms, logic bombs, etc. in English. However, to avoid the implicit indication of intent from the word malicious, the term *software weapon* is used in the paper presented at the 7th Nordic Workshop on Secure IT Systems (NordSec 2002), as well as in this thesis.

1.4 Scope

As stated in [3, 4] the taxonomy only covers software based weapons. This excludes chipping[3], which is regarded as being hardware based.

The work is not intended to be a complete coverage of the field. Due to a lack of good technical descriptions of software weapons, especially the empirical testing part of the thesis will not cover all different sectors of the field.

No other report or paper exclusively and in detail covering a taxonomy of software based IT weapons is known to have been published until now[4], but there are several simpler categorisation schemes of software weapons. Mainly they use two or three tiered hierarchies and concentrate on the replicating side of the spectrum, i.e. viruses and worms. They are all generally used as parts of taxonomies in other fields closely related to the software weapon field.

This affects the theoretic part of the thesis, which only describes some of the more recent and well known works in adjacent fields, containing parts formulating some kind of taxonomy or classification scheme of software weapons. These parts have also been evaluated to see how well they meet the requirements of a proper taxonomy.

Mainly the chosen background material covers taxonomies of software flaws, computer attacks, computer security incidents, and computer system intrusions.

1.5 Method

The method used for the research for this thesis has been concentrated on studies of other taxonomies in related computer security fields. Also more general information regarding trends in the development of new software weapons has been used. This has mainly been information regarding new types of viruses.

1.6 Intended readers

The intended readers of the thesis are those interested in computer security and the software based tools of information warfare. To fully understand the thesis the reader is recommended to read the paper presented at NordSec 2002 (see Appendix A) before reading the main text. The reader will also benefit from having some basic knowledge in computer security.

[3] Malicious alteration of computer hardware.

[4] This is of course as of the publishing date of this thesis.

1.7 Why read the NordSec paper?

This thesis rests heavily on a foundation formed by the NordSec paper, which is included as Appendix A. To really get anything out of the text in the thesis the paper has to be read before the thesis. In the following sections the reasons for this are further explained.

For those who have already read the paper, but need to refresh their memories, Section 2 contains the most important parts.

1.7.1 Chronology of work

The first outlines of the taxonomy were drawn in the summer of 2001, when the author was hired to collect different software based IT weapons and categorise them in some way. To structure the work a list of general characteristics of such weapons was made. Unfortunately the work with developing the list, which evolved into a taxonomy, took all the summer, so no weapons were actually collected. This ended in the publication of a report in Swedish [3] later the same year.

The presentation of the report was met with great interest and the decision to continue the work was taken. The goal was to get an English version of the report accepted at a conference, and NordSec 2002 was chosen. Once again the summer was used for writing and the result was positive, the paper got accepted.

However, before the answer from the NordSec reviewers had arrived, the decision was made that the paper-to-be was to be extended into a master's thesis. This work was set to start at the beginning of September 2002, at the same date as the possible acceptance from the NordSec reviewers was to arrive. The goal was to have completed the thesis before the beginning of 2003.

Therefore the work with attending to the reviewers comments on the paper, and the work on the master's thesis run in parallel, intertwined. The deadline for handing in the final version of the paper was set to the end of October. After that date the thesis work got somewhat more attention, until the NordSec presentation had to be prepared and then produced in early November. Finally all efforts could be put into writing the thesis.

The deadline for having a checkable copy of the thesis to present to the examiner was set to the end of November and therefore the decision to use the paper as an introduction was taken, to avoid having to repeat a lot of background material. Hence, due to a shortage of time in the writing phase of the thesis work and thus the paper being used as a prequel, the two texts are meant to be read in sequence. They may actually be seen as part 1 and 2 of the master's thesis.

1.7.2 Sequence of writing

As stated in the previous section the work with deepening the research ran in parallel with amending the paper text. Therefore the latest ideas and theories found were integrated into the paper text until the deadline. The subsequent results of the research was consequently put into the thesis.

Because the text was continuously written as the research went along, there was no time to make any major changes to the already written parts, as to incorporate them into the flow of the text. The alternative of cutting and pasting the paper text into the thesis was considered, but was regarded to take too much time from the work with putting the new results on paper. Hence, the text in the paper supplies the reader with a necessary background for reading the text in the thesis.

1.7.3 Line of thought

Because this taxonomy is the first one to deal with software weapons exclusively, the work has been characterised by an exploration of a not fully charted field. The ideas on how to best create a working taxonomy have shifted, but gradually settled down into the present form. Sometimes the changes have been almost radical, but they have always reflected the knowledge and ideas of that particular time. They therefore together span the field and thus are necessary to be acquainted with, because they explain why a certain solution was chosen and then maybe abandoned. Consequently, to be able to properly understand the taxonomy, how to use it, and follow the line of thought, the reader has to read both parts of the work, i.e. both the paper and the thesis.

1.8 Structure of the thesis

The thesis is arranged in five sections and three appendices that are shortly introduced below.

Section 1 This is the introduction to the thesis. It states the background and purpose of the thesis, together with some questions which will be answered in the document. Furthermore the scope, method, and intended readers are presented. There is also a subsection explaining why it is important to read the NordSec paper. Finally the structure of the thesis is outlined.

Section 2 To help those who have read the NordSec paper earlier to refresh their memories this section contains some of the more important parts of the paper. These include the reasons of why no other taxonomy of software weapons has been created, the discussion of the old and new definition of software weapons and a short introduction to the categories of the taxonomy as they were defined at that time.

Section 3 This section introduces the theories behind a proper taxonomy and also some reasons on why a taxonomy as this one is needed. The discussion, which was started in the NordSec paper on the problems regarding the use of the term *malware* (see Section 2.1.1), is continued.

Section 4 The section presents the evaluation of seven taxonomies from adjacent fields containing some sort of classification schemes of software weapons (called malware). Each evaluation shows how well the evaluated categorisation scheme meets the needs stated in Section 3.1.4 and the requirements stated in Section 3.2.2.1. The last part in the section summarises the evaluations.

Section 5 In this section the proposed taxonomy of software weapons is presented together with the accompanying definition. Each of the fifteen categories and their alternatives are discussed regarding changes, the reasons for including them in the taxonomy, and general methods to protect computer systems from weapons with such characteristics as the categories represent. The revisions made are mainly related to the formulation of the names of the categories and their alternatives. Also some of the categories have been extended to avoid ambiguity and make them exhaustive, and to facilitate a more detailed categorisation. Last in the section the result of a small test of the taxonomy is presented.

Section 6 This section contains the discussion part and the summary. Some general countermeasures to be used to secure computer systems from attacks are given. Also the future use and developments of the taxonomy needed to further push it towards a usable state are presented. Finally the thesis is summarised.

Appendix A This appendix contains the the paper presented at the NordSec workshop.

Appendix B In this appendix the categorisations of nine software weapons are given. The categorisations were made by the author of the thesis and are meant to function as a test of the taxonomy. The reader can independently categorise the same weapons as the author and then compare his or her results with the categorisations presented in this appendix.

Appendix C The appendix shows the proposed definitions (or categorisations), made by the author of the thesis, of the three terms *trojan horse*, *virus*, and *worm*. These categorisations indicates how the taxonomy may be used to redefine the nomenclature of the computer security field. Also completely new terms may be defined in this way.

2 The abridged NordSec paper

In this section some of the more important parts of the NordSec paper will be presented as they where published, to refreshen the memory of readers already familiar with the paper. To somewhat incorporate the text from the paper into the flow of the main text of the thesis, some changes of the layout and corrections of the spelling have been made. Apart from this everything else is quoted verbatim from the paper.

2.1 A Taxonomy of Software Weapons

My own hypothesis of why no other taxonomy of software weapons has yet been found can be summarised in the following points:

- The set of all software weapons is (at least in theory) infinite, because new combinations and strains are constantly evolving. Compared to the biological world, new mutations can be generated at light speed.
- It is hard to draw a line between administrative tools and software weapons. Thus it is hard to strictly define what a software weapon is.
- Often software weapons are a combination of other, atomic, software weapons. It is therefore difficult to unambiguously classify such a combined weapon.
- There is no unanimously accepted theoretical foundation to build a taxonomy on. For instance there are (at least) five different definitions of the term *worm* [5] and seven of *trojan horse* [6].
- By using the emotionally charged word *malicious* together with *intent*, the definitions have been crippled by the discussion whether to judge the programmer's or the user's intentions.

2.1.1 Preliminary Definition

The preliminary definition of software weapons[5] used at FOI[6] has the following wording (translated from Swedish):

> [...] software for logically influencing information and/or processes in IT systems in order to cause damage.[7]

[5] The term *'IT vapen'* (IT weapon) is used in the swedish FOI report.

[6] Swedish Defence Research Agency

[7] In Swedish: '[...] programvara för att logiskt påverka information och/eller processer i IT-system för att åstadkomma skada.'

This definition satisfies the conditions mentioned earlier in the text. One thing worth mentioning is that tools without any logical influence on information or processes are not classified as software weapons by this definition. This means that for instance a sniffer is not a software weapon. Even a denial of service weapon might not be regarded as a weapon depending on the interpretation of 'logically influencing ... processes'. A web browser on the other hand falls into the software weapon category, because it can be used in a dot-dot[8] attack on a web server and thus affect the attacked system logically.

Furthermore, the definition does not specify if it is the intention of the user or the programmer, that should constitute the (logical) influence causing damage. If it is the situation where the tool is used that decides whether the tool is a software weapon or not, theoretically all software ever produced can be classified as software weapons.

If instead it is the programmer's intentions that are decisive, the definition gives that the set of software weapons is a subset (if yet infinite) of the set of all possible software. But in this case we have to trust the programmer to give an honest answer (if we can figure out whom to ask) on what his or her intentions was.

A practical example of this dilemma is the software tool *SATAN*, which according to the creators was intended as a help for system administrators [7, 8]. *SATAN* is also regarded as a useful tool for penetrating computer systems [9]. Whether *SATAN* should be classified as a software weapon or not when using the FOI definition is therefore left to the reader to subjectively decide.

2.1.2 New Definition

When a computer system is attacked, the attacker uses all options available to get the intended result. This implies that even tools made only for administration of the computer system can be used. In other words there is a grey area with powerful administrative tools, which are hard to decide whether they should be classified as software weapons or not. Hence a good definition of software weapons is hard to make, but it might be done by using a mathematical wording and building from a foundation of measurable characteristics.

With the help of the conclusions drawn from the definitions of information warfare the following suggestion for a definition of software weapons was formulated:

> A software weapon is software containing *instructions* that are necessary and sufficient for a *successful attack* on a *computer system*.

[8] A dot-dot attack is performed by adding two dots directly after a URL in the address field of the web browser. If the attacked web server is not properly configured, this might give the attacker access to a higher level in the file structure on the server and in that way non-authorised rights in the system.

Even if the aim was to keep the definition as mathematical as possible, the natural language format might induce ambiguities. Therefore a few of the terms used will be further discussed in separate paragraphs.

Since it is a definition of *software* weapons, manual input of instructions is excluded.

2.1.2.1 Instructions

It is the instructions and algorithms the software is made of that should be evaluated, not the programmer's or the user's intentions. The instructions constituting a software weapon must also be of such dignity that they together actually will allow a breakage of the security of an attacked system.

2.1.2.2 Successful

There must be at least one computer system that is vulnerable to the tool used for an attack, for the tool to be classified as a software weapon. It is rather obvious that a weapon must have the ability to do harm (to break the computer security) to be called a weapon. Even if the vulnerability used by the tool might not yet exist in any working computer system, the weapon can still be regarded as a weapon, as long as there is a theoretically proved vulnerability that can be exploited.

2.1.2.3 Attack

An attack implies that a computer program in some way affects the *confidentiality*[9], *integrity*[10] or *availability*[11] of the attacked computer system. These three terms form the core of the continually discussed formulation of computer security. Until any of the suggested alternatives is generally accepted, the definition of attack will adhere to the core.

The security breach can for example be achieved through taking advantage of flaws in the attacked computer system, or by neutralising or circumventing its security functions in any way.

The term *flaw* used above is not unambiguously defined in the field of IT security. Carl E Landwehr gives the following definition [11, p. 2]:

> [...] a security flaw is a part of a program that can cause the system to violate its security requirements.

Another rather general, but yet functional, definition of ways of attacking computer systems is the definition of vulnerability and exposure [12] made by the CVE[12] Editorial Board.

[9] '[P]revention of unauthorised disclosure of information.'[10, p. 5]

[10] '[P]revention of unauthorised modification of information.'[10, p. 5]

[11] '[P]revention of unauthorised withholding of information or resources.'[10, p. 5]

[12] '[CVE is a] list of standardized names for vulnerabilities and other information security exposures – CVE aims to standardize the names for all publicly known vulnerabilities and security exposures. [...] The goal of CVE is to make it easier to share data across

2.1.2.4 Computer System

The term computer system embraces all kinds of (electronic)[13] machines that are programmable and all software and data they contain. It can be everything from integrated circuits to civil and military systems (including the networks connecting them).

2.2 A Draft for a Taxonomy

The categories of the taxonomy are independent and the alternatives of each category together form a partition of the category. It is possible to use several alternatives (where applicable) in a category at the same time. In this way even combined software weapons can be unambiguously classified. This model, called *characteristics structure*, is suggested by Daniel Lough [15, p. 152].

In Table 1 the 15 categories and their alternatives are presented. The alternatives are then explained in separate paragraphs.

Table 1. The taxonomic categories and their alternatives

Category	Alternative 1	Alternative 2	Alternative 3
Type	atomic	combined	
Affects	confidentiality	integrity	availability
Duration of effect	temporary	permanent	
Targeting	manual	autonomous	
Attack	immediate	conditional	
Functional area	local	remote	
Sphere of operation	host-based	network-based	
Used vulnerability	CVE/CAN	other method	none
Topology	single source	distributed source	
Target of attack	single	multiple	
Platform dependency	dependent	independent	
Signature of code	monomorphic	polymorphic	
Signature of attack	monomorphic	polymorphic	
Signature when passive	visible	stealth	
Signature when active	visible	stealth	

2.2.1 Type

This category is used to distinguish an *atomic* software weapon from a *combined* and the alternatives therefore cannot be used together.

A combined software weapon is built of more than one stand-alone (atomic or combined) weapon. Such a weapon can use more than one

separate vulnerability databases and security weapons.' [13]. The list is maintained by MITRE [14].

[13] This term might be to restrictive. Already advanced research is done in for example the areas of biological and quantum computers.

alternative of a category. Usage of only one alternative from each category does not necessarily implicate an atomic weapon. In those circumstances this category indicates what type of weapon it is.

2.2.2 Affects

At least one of the three elements *confidentiality*, *integrity* and *availability* has to be affected by a tool to make the tool a software weapon.

These three elements together form the core of most of the definitions of IT security that exist today. Many of the schemes propose extensions to the core, but few of them abandon it completely.

2.2.3 Duration of effect

This category states for how long the software weapon is affecting the attacked system. It is only the effect(s) the software weapon has on the system during the weapon's active phase that should be taken into account. If the effect of the software weapon ceases when the active phase is over, the duration of the effect is *temporary*, otherwise it is *permanent*.

Regarding an effect on the confidentiality of the attacked system, it can be temporary. If for example a software weapon e-mails confidential data to the attacker (or another unauthorised party), the duration of the effect is temporary. On the other hand, if the software weapon opens a back door into the system (and leaves it open), the effect is permanent.

2.2.4 Targeting

The target of an attack can either be selected *manual*[ly] by the user, or *autonomous*[ly] (usually randomly) by the software weapon. Typical examples of autonomously targeting software weapons are worms and viruses.

2.2.5 Attack

The attack can be done *immediate*[ly] or *conditional*[ly]. If the timing of the attack is not governed by any conditions in the software, the software weapon uses immediate attack.

2.2.6 Functional Area

If the weapon attacks its host computer, i.e. hardware directly connected to the processor running its instructions, it is a *local* weapon. If instead another physical entity is attacked, the weapon is *remote*.

The placement of the weapon on the host computer can be done either with the help of another, separate tool (including manual placement), or by the weapon itself. If the weapon establishes itself on the host computer

(i.e. breaks the host computer's security) it certainly is local, but can still be remote at the same time. A weapon which is placed on the host computer manually (or by another tool) need not be local.

2.2.7 Sphere of Operation

A weapon affecting network traffic in some way, for instance a traffic analyser, has a *network-based* operational area. A weapon affecting stationary data, for instance a weapon used to read password files, is *host-based*, even if the files are read over a network connection.

The definition of stationary data is data stored on a hard disk, in memory or on another type of physical storage media.

2.2.8 Used Vulnerability

The alternatives of this category are *CVE/CAN*[14], *other method* and *none*. When a weapon uses a vulnerability or exposure [12] stated in the CVE, the CVE/CAN name of the vulnerability should be given[15] as the alternative (if several, give all of them).

The alternative *other method* should be used with great discrimination and only if the flaw is not listed in the CVE, which then regularly must be checked to see if it has been updated with the new method. If so, the classification of the software weapon should be changed to the proper CVE/CAN name.

2.2.9 Topology

An attack can be done from one *single source* or several concurrent *distributed sources*. In other words, the category defines the number of concurrent processes used for the attack. The processes should be mutually coordinated and running on separate and independent computers. If the computers are clustered or in another way connected as to make them simulate a single entity, they should be regarded as one.

2.2.10 Target of Attack

This category is closely related to the category *topology* and has the alternatives *single* and *multiple*. As for the category topology, it is the number

[14]The term CAN (Candidate Number) indicates that the vulnerability or exposure is being investigated by the CVE Editorial Board for eventually receiving a CVE name [16].

[15]NIST (US National Institute of Standards and Technology) has initiated a meta-base called ICAT [17] based on the CVE list. This meta-base can be used to search for CVE/CAN names when classifying a software weapon.

The meta-base is described like this: 'ICAT is a fine-grained searchable index of standardized vulnerabilities that links users into publicly available vulnerability and patch information'. [18]

of involved entities that is important. A software weapon concurrently attacking several targets is consequently of the type multiple.

2.2.11 Platform Dependency

The category states whether the software weapon (the executable code) can run on one or several platforms and the alternatives are consequently *dependent* and *independent*.

2.2.12 Signature of Code

If a software weapon has functions for changing the signature of its code, it is *polymorphic*, otherwise it is *monomorphic*. The category should not be confused with *Signature when passive*.

2.2.13 Signature of Attack

A software weapon can sometimes vary the way an attack is carried out, for example perform an attack of a specific type, but in different ways, or use different attacks depending on the status of the attacked system. For instance a dot-dot attack can be done either by using two dots, or by using the sequence `%2e%2e`. If the weapon has the ability to vary the attack, the type of attack is *polymorphic*, otherwise it is *monomorphic*.

2.2.14 Signature When Passive

This category specifies whether the weapon is *visible* or uses any type of *stealth* when in a passive phase[16]. The stealth can for example be achieved by catching system interrupts, manipulating checksums or marking hard disk sectors as bad in the FAT (File Allocation Table).

2.2.15 Signature When Active

A software weapon can be using instructions to provide *stealth* during its active phase. The stealth can be achieved in different ways, but the purpose is to conceal the effect and execution of the weapon. For example man-in-the-middle or spoofing weapons use stealth techniques in their active phases through simulating uninterrupted network connections.

If the weapon is not using any stealth techniques, the weapon is *visible*.

[16] A passive phase is a part of the code constituting the software weapon where no functions performing an actual attack are executed.

3 Theory

The formulation of a taxonomy needs to follow the existing theories regarding the requirements of a proper taxonomy. They have evolved over time, but the core is more or less unchanged since Aristotle (384-322 B.C.) began to divide marine life into different classes [19, 20]. Some of the more recent works done within the computer security field dealing with taxonomies have also contributed to the theory.

A taxonomy also needs to be based on a good definition. This report discusses software weapons and consequently this term needs to be defined. One section therefore presents some alternative ways of defining software weapons, a.k.a malware.

3.1 Why do we need a taxonomy?

A field of research will benefit from a structured categorisation in many ways. In this section both general arguments for the use of a taxonomy, as well as more specific arguments concerning the computer security field, and specifically FOI, will be given.

3.1.1 In general

In [20] the main focus lies on the botanical and zoological taxonomies developed and used throughout time. In spite of this it gives a few general arguments for the use of a taxonomy. One of the main arguments is formulated in the following way:

> A formal classification provides the basis for a relatively uniform and internationally understood nomenclature, thereby simplifying cross-referencing and retrieval of information.

To enable systematic research in a field, there is a need for a common language and the development of a taxonomy is part of the formulation of such a language. [21] When searching for new things the history must first be known and understood. Therefore a common nomenclature within the field of research is vital, otherwise resent discoveries might not be remembered in a few years time and will have to be made again. This may lead to a waste of time and money.

Essentially a taxonomy summarises all the present knowledge within a field. In [22, p. 16] a citation from *The principles of classification and a classification of mammals* by George Gaylord Smith [23] with the following wording is presented:

> Taxonomy is at the same time the most elementary and the most inclusive part of zoology, most elementary because animals cannot be discussed or treated in a scientific way until some systematization has been achieved, and most inclusive

> because taxonomy in its various guises and branches eventually gathers together, utilizes, summarizes, and implements everything that is known about animals, whether morphological, physiological, or ecological.

The citation deals solely with zoology, but the idea is perfectly applicable to other fields as well, also computer security. There already exist frequently and commonly used terms for different types of software weapons. But they do not cover the complete field and thus do not help in structuring the knowledge attained this far.

A good taxonomy has both an *explanatory* and a *predictive* value. In other words, a taxonomy can be used to explain the scientific field it covers through the categorisation of entities. By forming groups, subgroups and so on with clear relationships in between, the field is easier to take in. The structuring also makes it possible to see which parts of the field that would benefit from more research. [22]

A parallel can be drawn to the exploration of a new world. To be able to find the unexplored areas, some knowledge of the ways of transport between the already explored parts will be of much help. Thus a structuring of the attained knowledge will speed up the exploration of the rest of the world.

A good and often used example of such a classification is the periodic system of elements. Simply by looking at the position of an element in the table, it is possible to get a feeling for the general properties of that element. The table has also been used to predict the existence of new elements, research which in the end has resulted in a couple of Nobel Prizes.

In [24, p. 21] the following arguments for the need of a categorisation are given:

- the formation and application of a taxonomy enforces a structured analysis of the field,
- a taxonomy facilitates education and further research because categories play a major role in the human cognitive process,
- categories which have no members but exist by virtue of symmetries or other patterns may point out white spots on the map of the field and
- if problems can be grouped in categories in which the same solutions apply, we can achieve more efficient problem solving than if every problem must be given a unique solution.

Therefore the scientists active within a field of research would gain a lot from spending some time and effort to develop a formally correct classification scheme of the field.

3.1.2 Computer security

Today none of the widely used terms given to different types of software weapons are strictly and unanimously defined. Almost every definition has some unique twist to it.

For example such terms as *trojan horse, virus,* and *worm* all have several different definitions for each term. Also the way the terms relate to each other differ among the classification schemes, as shown in Section 4. This is also described by Jakub Kaminski and Hamish O´Dea in the following way [25]:

> One of the trends we have been observing for some time now is the blurring of divisional lines between different types of malware. Classifying a newly discovered 'creature' as a virus, a worm, a Trojan or a security exploit becomes more difficult and anti-virus researchers spend a significant amount of their time discussing the proper classification of new viruses and Trojans.

Therefore some sort of common base to build a definition from is needed. If all terms used have the same base, they are also possible to compare and relate. By forming the base from general characteristics of software weapons the measurability requirement is met.

There is also a need for a better formal categorisation method regarding software weapons. By placing the different types of weapons in well defined categories the complete set of software weapons is easy to take in. Also the communication within the computer security community is facilitated in this way.

Much of the previous research being done has been concentrated to the three types of software weapons mentioned above. The concept of for example a denial of service (DoS) weapon was not on the agenda until the large attacks on eBay, Yahoo and E*trade took place. Because these weapons represents rather new concepts, they sometimes are forgotten when talking about software weapons. This is unfortunate, because in a study done in 2001 the number of DoS attacks on different hosts on the Internet over a three week period was estimated to be more than 12,000. [26] A categorisation of the complete set of software weapons would consequently lessen the risk of forgetting any potential threats.

The research in computer security would also benefit from having a common database containing specimens of all known software weapons. Both the problem with naming new software weapons and the tracing of their relationship may be solved having access to such a database.

Another thinkable field of use is in forensics. In the same way as the police have collections of different (physical) weapons used in crimes today, they (or any applicable party) may benefit from having a similar collection of software weapons. Then the traces left in the log files after an attack may be used as unique identifiers to be compared to those stored in the

software weapon collection. If needed the weapon may even be retrieved from the collection and used to generate traces in a controlled environment.

Today many anti-virus companies maintain their own reference databases for computer viruses, but there is no publicly shared database. Therefore the WildList Organization International has taken on the challenge of creating such a database for computer viruses. [27]

3.1.3 FOI

Regarding the specific needs for a taxonomy at FOI, they mainly relate to defensive actions and the protection of military computer systems. For example there is a need for tools to help creating computer system intrusion scenarios. [28] One part of such a tool would be some sort of rather detailed descriptions of the general characteristics of different existing and also non-existing, but probable, software weapons. These descriptions therefore need to be both realistic regarding the weapons existing today, as well as comprehensive enough to be usable even in the foreseeable future.

The threats posed to the network centric warfare (NCW) concept by different software weapons have to be met. To be able to do that the properties of different types of weapons have to be well structured and well known to make it possible to counter them in an effective way.

The level of detail of the categorisation needs to be rather high, but yet usable even by laymen. Therefore also the used vocabulary (for example the names of the different classes) need to be both general and technically strict.

There is also a need to extend the terminology further, especially in the non-viral software weapon field. There are as many different types of viruses defined as there are of all other software weapons together. For example in [29] fourteen different types of viruses and ten non-viral weapons are listed. And in [30] there are eleven non-viral software weapons given and about as many types of viruses (depending on how they are categorised). In [31] two (three including joke programs) types of non-viral software weapons and five or ten virus types (depending on the chosen base for the categorisation) are presented.

To facilitate the creation of the scenario tools mentioned above many more types of software weapons are needed than what the categorisation schemes offer today. What really is needed is the same level of detail as offered by the scenario creation tools used for conventional warfare. These tools sometimes contains classes of troop formations down to platoon level.

In a computer system intrusion situation (not only directly involving the military) all involved personnel need to be fully aware of what the different terms used really mean. Thus the terminology needs to be generally accepted and unambiguous. To enable the definition of such generally accepted terms some common base has to be used. A natural base to build a

definition from would be the technical characteristics of the weapons representing the different terms.

A taxonomy of software weapons will have educational purposes too, especially when training new computer security officers. Then the usability of the taxonomy is very important. Each category and its alternatives need to be easy to understand and differentiate. The taxonomy then also may function as an introduction to the different technologies used in the software weapon world.

Because of the intended use in the development of the defence of military computer systems, the categories have to be defined as unambiguously as possible. They also have to be measurable in some way, to enable the objective evaluation of the defensive capacity of different proposed computer security solutions.

3.1.4 Summary of needs

The different reasons for having a taxonomy of software weapons can be summarised in the following points:

- The nomenclature within the computer security field needs to be defined in an objective, generally accepted, and measurable way, because today the lines between the terms are blurring. It also has to be further extended, especially within the non-viral field.
- The use of a taxonomy makes a structured analysis and thus a more scientific approach to the software weapon field possible. In that way the field will be easier to take in, which would benefit the training of new computer security personnel. Also the future research will be helped by the predictive properties of a taxonomy.
- To be able to find better solutions to problems quicker and lessen the risk of forgetting important types of weapons a good way of grouping different software weapons is needed.
- When constructing computer system intrusion scenarios a rather detailed categorisation of the different tools available, both today and in the future, is needed.

3.2 Taxonomic theory

In this section the theory behind a taxonomy will be presented. First of all the classical theory dating back to Aristotle (384–322 B.C.) is introduced. Then the formal requirements of a taxonomy are specified and connected to the need for a taxonomy of software weapons. Finally some of the taxonomies in the computer security field are evaluated with respect to how well they fit the requirements of a taxonomy of software weapons. The evaluated taxonomies were chosen because they were well known, closely related to the software weapon field, and fairly recently written.

3.2.1 Before computers

The word *taxonomy* comes from the Greek words *taxis* (arrangement, order) and *nomos* (distribution) and is defined in the following way in [32]:

> Classification, esp. in relation to its general laws or principles; that department of science, or of a particular science or subject, which consists in or relates to classification; esp. the systematic classification of living organisms.

Another definition of the term *taxonomy*, this time from a more explicit biological point of view, is given in [33]:

> [SYSTEMATICS] A study aimed at producing a hierarchical system of classification of organisms which best reflects the totality of similarities and differences.

In the beginning the word was used in zoology and botany, but in more recent times the usage has been widened and today comprises almost every thinkable field. This trend has actually started to make the term somewhat watered down, which is unfortunate. In many cases the taxonomies are simply lists of terms, lacking much of the basic requirements of a taxonomy stated in the theory.

The fundamental idea of a taxonomy is described in the following way in [11, p. 3]:

> A taxonomy is not simply a neutral structure for categorizing specimens. It implicitly embodies a theory of the universe from which those specimens are drawn. It defines what data are to be recorded and how like and unlike specimens are to be distinguished.

According to Encyclopedia Britannica the American evolutionist Ernst Mayr has said that ‘taxonomy is the theory and practice of classifying organisms’. [20] This quotation summarises the core of the ideas behind a taxonomy in a good way.

The first one to look into the theory of taxonomies was Aristotle. He studied the marine life intensively and grouped different living things together by their nature, not by their resemblance. This form of classification was used until the 19th century. [19, 20]

In 1758 the famous Swedish botanist and zoologist Carolus Linnaeus (Carl von Linné), usually regarded as the father of modern taxonomy, used the Aristotelian taxonomic system in his work. He extended the number of levels in the binomial hierarchy and defined them as class, order, genus, and species. In other words, he should really not be credited for inventing the taxonomy, but for his work in naming a big amount of plants and animals and creating workable keys for how to identify them from his books. [20]

When Darwin in 1859 published his work ‘The Origin of Species’ the theory of taxonomy began to develop and seep into other fields. [22] Later both Ludwig Wittgenstein and Eleanor Rosch have questioned the theory. The work of Rosch led to her formulation of the *prototype theory*, which suggests that the categories of a taxonomy should have prototypes against which new members of the category are compared. [24]

The idea of having a prototype to compare new members against is also stated in [20]. Such prototypes should be stored in a public institution, so researchers can have free access to the material. It is then also possible to correct mistakes made in earlier classifications, the first taxonomist maybe missed an important property, or new technology makes it possible to further examine the prototype.

Additionally, by having one publicly available specimen being the criterion of the group, it is in reality working as a three dimensional, touchable definition of the members of the group.

There is also a third theory mentioned in [24] and that is *conceptual clustering*. The theory is by some regarded as lying between the classical theory and prototype theory. In short it states that items should be arranged by simple concepts instead of solely on predefined measures of similarity. The theory is directed towards automatic categorisation and machine learning.

3.2.2 Requirements of a taxonomy

Some of the references used in this section relates to biology, others to computer security. The given references and requirements are really applicable to all types of taxonomies and thus also to a taxonomy of software weapons.

To make a taxonomy usable in practice, it must fulfil some basic requirements. First of all, a taxonomy without a proper purpose is of little or no use and thus the purpose must be used as a base when developing the taxonomy. To fit the purpose the items categorised with the help of the taxonomy must be chosen in some way. Therefore the taxonomy has to be used in conjunction with a definition of the entities forming the field to be categorised, because the definition functions as a filter, which excludes all entities not belonging to the field and thus not fitting the taxonomy. How to formulate such a definition for software weapons is further discussed in Section 3.3.

Also, the properties of the items to be categorised, i.e. the categories of the taxonomy, must be easily and objectively observable and measurable. If not, the categorisation of an item is based on the personal knowledge of the user of the taxonomy, as stated in this citation from [22, p. 18]:

> Objectivity implies that the property must be identified from the object known and not from the subject knowing. [. . .] Objective and observable properties simplify the work of the

> taxonomist and provide a basis for the repeatability of the classification.

In [15, p. 38] a list compiled from five taxonomies in different fields of computer security is presented. From that list four different properties can be extracted that the categories of a taxonomy must have. These properties are stated in [21, 22, 24, 34, 35], although different names are used in some papers. The categories must be:

- mutually exclusive,
- exhaustive,
- unambiguous, and
- useful.

If the categories are not mutually exclusive the classification of an item cannot be made, because there are more than one alternative to choose from. This property is closely connected to the property *unambiguous*. If a category is not clearly defined and objectively measurable, the boundary between different categories becomes inexact and an item may belong to more than one category.

The property *exhaustive* is also important. If the category does not completely cover all possible variations in the field, an entity may be impossible to categorise. It simply does not belong anywhere, even if it should. Thus an alternative *other* may be needed to make a category exhaustive, although then there is a risk of getting too many entities categorised in this class.

Finally the categories have to be useful, which is connected to the whole idea of having a taxonomy. As mentioned in the beginning of this section a taxonomy must have a purpose to be of any use. In [24, p. 85] it is stated that:

> The taxonomy should be comprehensible and useful not only to experts in security but also to users and administrators with less knowledge and experience of security.

Even Lough mentions the usefulness as an important property [15, p. 2]. If the categories and terminology used in the taxonomy are hard to understand, the group of people able to use it tend to be rather small and the descriptive property is lost.

3.2.2.1 Summary of properties

If the categories of a taxonomy lack any of the properties mentioned in this section, a classification done by one person cannot be repeated by another, or even by the same person at different occasions. Then, in practice, the taxonomy becomes useless. Therefore, the approach taken in this thesis is that a proper taxonomy is required to:

- have a definition *properly limiting* the set of items to be categorised,
- have categories based on *measurable properties* of the items to be categorised,
- have *mutually exclusive* categories,
- have *exhaustive* categories,
- have *unambiguous* categories, and
- be formulated in a language and way that makes it *useful*.

3.3 Definition of malware

How to define malware (or whatever name used) is a disputed question. Most, if not all the different definitions previously made incorporate malicious *intent* in some way. The problem is that it is very hard, if not to say impossible, to correctly decide the intent behind the creation or use of a software. The problem is described in the following way in [36], which is quoted verbatim:

> Dr. Ford has a program on his virus testing machine called qf.com. qf.com will format the hard drive of the machine it is executed on, and place a valid Master Boot Record and Partition Table on the machine. It displays no output, requests no user input, and exists as part of the automatic configuration scripts on the machine, allowing quick and easy restoration of a "known" state of the machine. Clearly, this is not malware.
>
> 1. If I take the executable, and give it to my wife, and tell her what it is, is it malware?
> 2. If I take the executable, and give it to my wife, and don't tell her what it is, is it malware?
> 3. If I mail the executable to my wife, and tell her it is a screen saver, is it malware?
> 4. If I post the executable to a newsgroup unlabelled[,] is it malware?
> 5. If I post the executable to a newsgroup and label it as a screensaver[,] is it malware?

Ford then concludes that the only thing not changing is the software itself. Therefore his personal belief is '[. . .] that any definition of malware must address what the program is expected to do'. But he does not specify what he means with 'expected to do'. Is it up to each user to decide what is expected? Or should it be more generally defined expectations, and if so, how should they be found?

Another article debating the use of the words malicious intent in the definition of malware is written by Morton Swimmer [37]. There he states that:

> In order to detect Malware, we need to define a measurable property, with which we can detect it. [...] "Trojan horses" are hard to pin one particular property to. In general, "intent" is hard even for a human to identify and is impossible to measure, but malicious intent is what makes code a Trojan horse.

He gives viruses the property of self-replication, but then argues that the consequence of such a definition is that a copy program copying itself would fit the definition, and thus be a virus. In other words, a false positive[17].

Also there will always be false negatives[18], in [38] Fred Cohen mathematically prove that the same definition as above of the virus property is undecidable. The proof is built on the idea that a possible virus has the ability to recognise whether it is being scanned by a virus detection algorithm looking for the virus property. If the virus detects the scanning, it does not replicate, it just exits, i.e. it is not a virus in that context. The virus code would look something like this[19]:

```
if (Scan(this) == TRUE) {
   exit();
} else {
   Replicate(this);
}
```

Cohen's proof has been criticised for being too theoretical, and only valid in a rather narrow environment. A generalisation of the original proof has been presented by three scientists at IBM in [39].

Also Kaminski and O´Dea have commented on the problem of determining whether a tool is malicious or not. They write, in the abstract of a paper [25] presented at the Virus Bulletin 2002 conference, that:

> [...] the real problems start when the most important division line dissolves - the one between intentionally malicious programs and the legitimate clean programs.

As can be deducted from above, the use of intent in the definition of malware is not optimal, because it is impossible to measure. If the creator of a software tool is found, it is very hard to decide if he or she gives an honest

[17] An indication of something being of some type, which it in reality is not. Crying 'wolf!' when there is none, so to speak.

[18] Something in reality being of some type, which it is not indicated as being of. Saying 'lamb', when one really ought to cry 'wolf!' instead.

[19] The Java-like code might very well be optimised, but it has not been done because of readability issues.

answer to the question on the intended purpose of the software tool used in an attack.

Consequently a new way of defining a software weapon has to be found, a definition not involving intent in any way. It has to be based on a measurable property of a software weapon and focus on the weapon itself, not the surrounding context or other related things.

Therefore the following formulation of a definition is proposed to be used in conjunction with the taxonomy [4]:

> A software weapon is software containing *instructions* that are necessary and sufficient for a *successful attack* on a *computer system*.

This definition is further explained in Section 2.1.2 and also in Section 5.1.

4 Earlier malware categorisations

Although the concept of a categorisation of the existing software weapons has been proposed a few times already, nobody has yet dedicated a whole paper to it. In this section some of the works containing some kind of proposed categorisation of software weapons are presented. Each presentation is followed by a short evaluation of its significance and how well it meets the requirements of a taxonomy of software weapons. Each summary following an evaluation includes a figure showing how the specific taxonomy relates the three terms trojan horse, virus and worm to each other.

4.1 Boney

The purpose of this paper is to develop a software architecture for offensive information warfare. [40] Thus Boney needs to form a taxonomy from earlier work in rogue programs, which are defined as all classes of malicious code. He credits Lance Hoffman[20] for inventing the term and follows the discussion in a book written by Feudo[21]. Boney writes that rogue programs primarily have been used in denial of service attacks.

He lists trojan horses, logic bombs, time bombs, viruses, worms, trapdoors and backdoors as being the complete set of malicious programs. His definition of a trojan horse states that it is appearing as a legitimate program and at the same time performing hidden malicious actions. A virus in its turn '[...] may be a trojan horse but has the additional characteristic that it is able to replicate'. [40, p. 6] The more formal definition of a virus states that it is parasitic and replicates in an at least semi-automatic way. When transmitting itself it uses a host program. Worms are defined as being able to replicate and spread independently through network connections. If they too may be trojan horses is not explicitly stated by Boney, but he writes that the difference between a virus and a worm is the way they replicate. Eventually the conclusion may be drawn that also worms might be trojan horses.

4.1.1 Summary of evaluation

Boney's taxonomy is rather simple, it is a number of short definitions of some common terms used in the computer security field. He does not mention if the list is meant to be exhaustive.

He states that '[a] virus may be a Trojan horse' [40, p. 6], but at the same time he does not define virus as a subclass of trojan horse, which indicates that the two categories are not mutually exclusive. The same

[20]The book is not part of the background material used for this thesis. If anyone is interested the reference to the book is [41].

[21]The book is not part of the background material used for this thesis. If anyone is interested the reference to the book is [42].

thing may be true for worms, but Boney does not explicitly state whether worms may be trojan horses (see Figure 1).

Consequently the categorisation scheme does not fulfil the requirements of a taxonomy stated in this thesis (see Section 3.2.2.1). Also the shortness and lack of clear definitions make the taxonomy not fulfilling the needs of FOI for a detailed taxonomy (see Section 3.1.3).

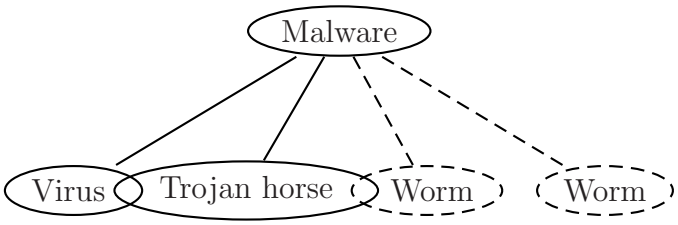


Figure 1. The relationship of a trojan horse, a virus and a worm according to Boney.

4.2 Bontchev

The report does not give any specific definition of the term malware, more than referring to it as ‘malicious computer programs’. The goal of the presented classification scheme is to make it cover all known kinds of malicious software [30, p. 11].

Four main types of malware are given; *logic bomb*, *trojan horse*, *virus*, and *worm*. They are then further divided into sub-categories. The relationship between the different types of malware are given implicitly by the levels of the section headers used in the report. [30, pp. 14–22]

Logic bomb is the simplest form of malicious code and can be part of other types of malware, often trojan horses. A special variant of a logic bomb is a *time bomb*. Logic bombs typically work as the triggering part of other types of malicious software.

Trojan Horse is defined as a piece of software containing one or more, by the user, unknown and destructive functions. Often the trojan horse also poses as a legitimate software. If the software warns the user and asks for authorisation when the destructive function is activated, it is *not* a trojan horse.

Virus is described as a computer program that is able to replicate by attaching itself to other computer programs in some way. The program the virus attaches to is called a *host* or *victim* program.

Worm is a replicating stand-alone program, which in some cases can be regarded as a subclass of viruses, according to Bontchev.

Bontchev writes that most specialists favour the view that viruses are not to be regarded as forming a subclass of trojan horses. Instead the two types are to be placed on the same level, with viruses defined as replicating software and trojan horses are non-replicating. His definition of a trojan

horse (as shown above) only specifies that there should exist destructive functions, unknown to the user, and that there should be no warning when the destructive function is activated. He does not explain why he chose not to follow the other experts.

A subclass of the worm class is the software weapon type *chain letter*, which is defined as the simplest form of a worm. It is an attachment to an e-mail and needs user intervention to be able to execute and replicate. The text part of the message is meant to convince the user the attached file contains some useful function. But instead the weapon performs some kind of destructive action, mostly including sending the message and attachment on to addresses found in the affected user's address book. Consequently this description does fit both the worm class and the trojan horse class, but Bontchev does not mention this, or tries to solve the ambiguity.

4.2.1 Summary of evaluation

Bontchev states that worms sometimes can be considered a special case of viruses. This makes the resulting tree somewhat difficult to draw. However, one possible variant is the one shown in Figure 2.

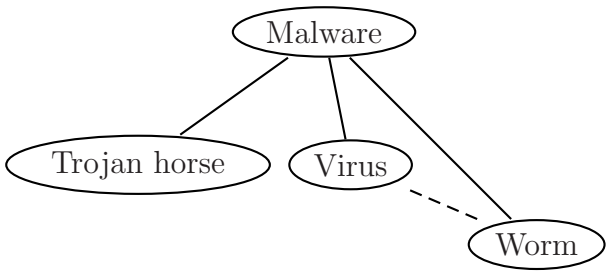


Figure 2. The relationship of a trojan horse, a virus and a worm according to Bontchev.

His definitions of the non-viral software weapons are not completely mutually exclusive in some cases. One example is the definition of chain letters, which also fits the definition of trojan horses.

His own definition of trojan horses is contradicted when he writes that the view favoured by most specialists is the division of viruses and trojan horses into replicating respectively non-replicating programs. He does not give any reason for his choice to not follow the other specialists. This unfortunately brings some ambiguity to his work.

An example of the taxonomy not being exhaustive (and at the same time ambiguous) is the logic bomb, which is said to most often be embedded in larger programs and there be used to trigger for instance a trojan horse. But if the logic bomb resides inside another program, it may be viewed as the unknown and destructive function defining a trojan horse. Thus, the definition of the logic bomb as a separate class does really necessitate the forming of other types of hidden and destructive functions being part of trojan horses.

Even if the part dealing with viruses is rather detailed, the taxonomy as a whole is too coarse to really fit the needs for a detailed taxonomy

stated in Section 3.1.4. Nor are the formal requirements, formulated in Section 3.2.2.1, fulfilled.

4.3 Brunnstein

In [43] Klaus Brunnstein writes about the difficulties of defining malware. He regards the traditional definitions as self-contradicting and not exhaustive. Therefore he proposes a new way of defining the term, which he calls *intentionally dysfunctional software*. His definition is meant to distinguish *normal* dysfunctionalities from *intentionally malevolent* ones.

To be able to define the term, he postulates that all software which is *essential* to some business or individual also is governed by a specification of all its functions (at least those which may have an effect on the system in use). If not, such a specification can be replaced by some sort of reverse engineering.

He then defines *functionality* in the following way [43, Def. 1–2] (quoted verbatim from the source):

> *A program´s or module´s or object‘s ”functionality”* is characterized by the set of all specifications, formal or informal, from which information about ”proper work” of a program can be concluded, and from which certain undesired functions can be excluded.
>
> Remark: it is irrelevant whether the manufacturer‘s specifications, formal or informal, are explicitly known to the user. Even if a manufacturer decides to hide functions (e.g. for objects with limited visibility and inheritance), such functions belong to the functionality of a software. If a manufacturer decides to include some hidden Trojanic payload, then this becomes part of the specification and therefore the functionality of that software.
>
> [. . .] *A software or module is called ”dysfunctional”* when at least one function deviates from the specification.

In other words, if the creator of a software weapon includes the destructive functions in some sort of secret specification, the software is perfectly good (or not dysfunctional anyway). He also admits this consequence later in the text, at least regarding trojan horses.

According to Brunnstein, intentionally dysfunctional software is a piece of code where some essential function is not contained in the manufacturer's specification [43, Def. 3]. He also writes that the deviation from the specification shall be *significant* to make the software dysfunctional. Later he states that [43, Def. 4]:

> *A software or module is called ”malicious” (”malware”)* if it is *intentionally dysfunctional*, and if there is sufficient evidence (e.g. by observation of behaviour at execution time)

> that dysfunctions may adversely influence the usage or the behaviour of the original software.

It is left to the reader to decide what 'essential function' and 'significant deviation' really mean. Neither does he try to grade these ambiguous terms to make the definitions easier to use.

He continues his line of argument with the definition stating how software is turned into malware. The text is quoted from [43, Def. 5]:

> *A software or module* with *given functionality* is transformed into ”malware” by a process called *”contamination”*.

The definition gives that it is not the contaminating code that should be regarded as malware, but the victim of the contamination.

Brunnstein then gives three types of contamination: *infection*, *propagation* and *trojanisation*. [43, Example of def. 5] The first two relates to viruses and worms respectively and the last one, logically, to trojan horses. He then defines a trojan horse in the following way, quoted from [43, Def. 7]:

> A ”Trojan Horse” is a software or module that, in addition to its specified functions, has one or more additional hidden functions (called ”Trojanic functions”) that are added to a given module in a contamination process (”trojanization”) usually unobservable for a user. These hidden functions may activate depending upon specific (trigger) conditions.

However, it is somewhat unclear if the definition should be interpreted as trojan horses having an infectious property, or if it is the victim of a trojanisation that becomes a trojan horse. The definition of contamination stated in [43, Def. 5] gives that the latter alternative probably is the correct one.

To handle the software working as specified, but having intentionally destructive functions, he introduces a new term; *critter*. However, such software is not to be included in the malware category, according to him.

Brunnstein writes that *real malware* '[...] may be constructed by repetitively combining different types or instances of self-reproducing software for one or several platforms with Trojanic functions' and gives an example in WNT/RemoteExplorer.

Finally he summarises his line of thought in a final definition of how malware may appear [43, Def. 8]:

> Malware may be developed from a given (functional) software or module by intentionally contaminating it with unspecified (hidden) functions. Such malware may consist of combinations of self-replicating or propagating part, or both, which may be triggered by some built-in condition. Malware may include hidden Trojanic functions, which may also activate upon

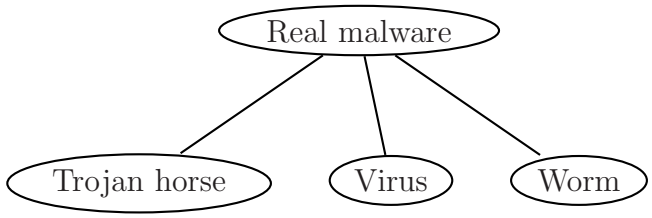
> some built-in condition (trigger). The development of malware (in the contamination process, namely the Trojanization) may be observed in cases of self-reproducing software, but it is (at present) difficult to anticipate the malicious Trojanic behaviour before it materializes.

He claims that by using these definitions it is possible to completely characterise all currently known malwares by their combinations of replicative and trojanic parts.

4.3.1 Summary of evaluation

Brunnstein does not present a real hierarchical system. Instead he concentrates on the definition of malware and therefore really has only one level in his hierarchy. This level contains three types of malware, namely trojan horse, virus and worm, which then are combined into what he calls 'real malware'. This is shown in Figure 3.

Figure 3. The relationship of a trojan horse, a virus and a worm according to Brunnstein.



The goal of creating a definition distinguishing normal software dysfunctionalities from intentionally malevolent ones, which Brunnstein stated in the paper, is not reached. By concentrating on the *specification* of software he misses all those softwares which are intended and specified to have the ability to create havoc in computer systems. Such softwares, given the name *critters*, are explicitly said not to be malware.

Another problem with the proposed definitions is the idea that a malware is formed in a *contamination* process. Brunnstein states that a good software is transformed into malware by being contaminated with non-specified functions that may adversely affect the usage of the system. The definition might work if applied in a software development environment, but not as it is now, on real and existing software, which has passed the developmental phase. The effect is that what Brunnstein defines as malware really is the victim of for instance a virus attack. What he does may be compared to trying to eradicate a disease by declaring the patients as evil. Of course, if it is possible to kill the patients faster than the disease can infect new victims, the battle might be won. The question is, who won the war?

His declaration of *real malware* as being a combination of trojan horses, viruses, and worms may have the effect that almost all existing malwares will belong to the same category. There is a maximum of seven different

categories to place a specific malware in and the present trend is to create more and more complex combinations of malware from simpler types. Consequently there is a risk of getting almost all malwares in a single category, namely the trojan horse-virus-worm one.

The rather ambiguous vocabulary used for the definitions and the fact that all malwares are seen as contaminated, makes the proposed definitions and classification scheme not fulfilling the needs stated in Section 3.1.4. Nor are the requirements specified in Section 3.2.2.1 fulfilled.

4.4 CARO

All the anti-virus companies today name the viruses they have found in their own way, several of them similar to the Computer Antivirus Research Organization (CARO) naming convention established in 1991. [29, 44] Unfortunately the companies have not managed to agree on a common implementation. One of the bigger obstacles on the road towards a common naming scheme is money. Each company wants to show that they where the first to find a virus and also to present a cure for it. Therefore they are reluctant to share information to facilitate a common naming of a virus. [45]

An attempt to fix this has been made. The project is named VGrep and is a list or database linking the different names used by the anti-virus companies to the same virus. More information can be found at [46].

CARO is, as written above, a naming convention and should not be evaluated as a taxonomy. However, one of the reasons for using a taxonomy is to be able to name the entities in the field in question and in that way get a better understanding of them. The CARO naming scheme also divides viruses into a four (or actually five) tiered hierarchy and thus have some resemblance of a taxonomy. The levels are [47]:

1. family name
2. group name
3. major variant
4. minor variant
5. modifier

The authors propose an informal way of grouping the different existing viruses into families, by categorising them after their structural similarities. For example small viruses which only replicate and do not contain any other distinctive feature are grouped into six families depending on the type of file they infect. The given list is not exhaustive any longer, because it only states that `.COM` or `.EXE` files are infected. There are no alternatives for the other types of executable files (or really interpreted, for instance Java) used by more recent viruses.

The lower levels are defined in similar ways. Most parts of the definitions deal with how to form a proper name, which words are to be used and not to be used.

Scheidl proposes in [48] an extension to the CARO naming convention adding categories for *platform, multi-partite virus, type, and language*. The category *type* does specify other types of software weapons. The new types are [48, p. 2]:

Joke – just a funny program made to pull someone's leg, not a virus.

Testfile – an anti-virus test file such as the EICAR-testfile.

Trojan – a program which claims to be useful but turns out to be malware at some point during the execution.

Worm – a program which does not replicate on the same computer as it resides on, but spreads over networks.

Dropper – not a virus, but a program that drops a virus.

Germ – the first generation of a virus in its initial, programmed form.

Intended – a program which is intended to be a virus, but which for some reason cannot replicate.

Malware – an unspecified type of malware.

4.4.1 Summary of evaluation

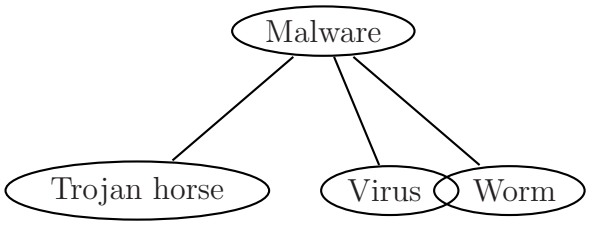
The CARO naming convention is specifically stated to be a naming scheme by the authors and therefore should not really be treated as a taxonomy. However, it is a rather significant document and it does build on categorising viruses into families, groups, etc. It is included in the evaluation because it might be possible to use it to categorise software weapons anyway.

The original version of the naming convention does only cover viruses and it lacks a category for file viruses infecting other file types than `.COM` or `.EXE` files. This makes the convention not fulfilling the requirement of a taxonomy to have exhaustive categories. Neither version defines the term *virus* and it is thus hard to decide whether the proposed extension makes the categories exhaustive, even if the extension adds more file types.

The proposed extension made by Scheidl does have categories for trojan horses and worms. They are put on the same level as viruses, but a virus with the abilities of both a worm and a virus is to be classified as a virus (the term virus is not defined anywhere in the text). Therefore the two classes are not mutually exclusive, as shown in Figure 4.

The above mentioned reasons implies that the CARO naming convention, with or without the extension by Scheidl, does not fulfil the needs specified in Section 3.1.4, or the requirements of a proper taxonomy stated

Figure 4. The relationship of a trojan horse, a virus and a worm according to Scheidl.



in Section 3.2.2.1. It therefore is hard to use as a basis to build a complete taxonomy from, without major changes being made to the scheme.

4.5 Helenius

Helenius has written a dissertation with the title '*A System to Support the Analysis of Antivirus Products' Virus Detection Capabilities*'. [31] Hence, the reason for having a classification of malware (here called harmful program code) is to familiarise the reader with certain terms used in the dissertation. Helenius also needs to establish a set of terms describing the different types of harmful program code handled by the anti-virus products. Thus, a classification scheme of harmful program code is formulated and also two ways of categorising viruses, one based on the infection mechanism and one based on more general characteristics.

Helenius first concludes that not even among computer anti-virus researchers the term *malware* is unanimously agreed on. He also points out that the term is hard to define because the maliciousness of a software depends on the purpose of the use and gives the example of the disk formating tool presented in [36] (see also Section 3.3).

4.5.1 Harmful program code

The classification scheme of harmful program code he presents is constructed from Brunnstein's definition (see Section 4.3). However, this scheme is not as detailed as Brunnstein's and also differs in some ways. He has also been influenced by Bontchev (see Section 4.2). This can be seen from Helenius definition of harmful program code as being '[...] any part of a program code which adds any sort of functionality against the specification or intention of the system'. [31, p. 12]

He then continues by stating that '[h]armful program code includes all program parts which are against the system's specification or intention' [31, p. 12]. However, he does not specify how the intention of the system is to be measured or whom to ask.

The interesting part of the scheme (from the point of view of this thesis) is the part defining intentionally harmful program code, which is said to be equal to malicious program code. The class includes four types; trojan horses, computer viruses, joke programs and malicious toolkits (in the accompanying figure there is a fifth type; others). Helenius admits the list

may not be exhaustive[22].

The category *joke programs* is defined in the following way by Helenius [31, p. 12]:

> [...] a program which imitates harmful operation, but does not actually accomplish the object of imitation and does not contain any other malicious operation.

Noteworthy is the fact that they are regarded as only *imitating* harmful operations, without accomplishing anything harmful. Helenius does not further explain the underlying causes for including them in the intentionally harmful program code class.

Computer viruses are said to have the capability to replicate recursively by themselves and may also include operations typical for trojan horses and malicious toolkits. However, this does not make them belong to those categories, according to Helenius.

The same thing is said to be valid for computer worms, but they are instead *independent,* by themselves recursively replicating programs. He also specifies them as a subgroup of computer viruses.

He defines a trojan horse as a self-standing program with hidden destructive functions, in the same way as Bontchev does (see Section 4.2). The term *self-standing* is said to have the meaning *not being able to replicate by itself.* In the same way as for the types described above he writes that a trojan horse might include operations typical for a malicious toolkit, but that does not make the trojan horse belong to that category.

Finally Helenius describes a malicious toolkit, which is said to be designed to help malicious intentions aimed at computer systems. The class includes such programs as virus creation toolkits, among others.

4.5.2 Virus by infection mechanism

Helenius divides computer viruses into $4 + 1$ groups based on their infection mechanisms. Four groups are mutually exclusive and the fifth group indicates that two or more of the mechanisms are used together in the virus. The groups are:

File viruses, which are viruses replicating via infecting executable files.

[22] The actual wording used in Helenius dissertation is: 'The list may not be exclusive.' [31, p. 12] This has been regarded a typing error. Helenius' text can be interpreted in two ways, either there is a word missing (*mutually*), or he really meant to write *exhaustive*. Because Helenius specifically writes that '[a malware type] may include operations, which are typical for [other types of malware], but this does not make such [types into other types]' the list actually becomes mutually exclusive. Thus he probably did not intend to write 'may not be [mutually] exclusive'. Furthermore, in the figure accompanying the scheme in the dissertation there is an extra category named 'Others?', which makes the class exhaustive. This category is not included in the text and therefore the more probable alternative is that he meant to write exhaustive.

Boot sector viruses, which replicate by infecting boot sectors of diskettes or hard disks, or partition sectors of hard disks, or a combination thereof.

Macro viruses, which use application macros for replication.

Script viruses, which replicates via operating scripting language, such as for example DOS batch files, Visual Basic Scripting, or Unix shell scripts.

Multi-partition viruses, which form a combination of two or more of the previous four infection mechanisms.

However, in the classification scheme of harmful program code, worms are said to be a subclass of viruses, but that is not reflected in this scheme. Furthermore, worms are said to be independent programs capable of replicating on their own, without using a host program. Therefore not all the different mechanisms in the scheme are applicable to worms. This is especially true for the file virus class, an independent program using a host program is a contradiction.

4.5.3 Virus by general characteristics

The classification by characteristics is shown as a tree, but would fit equally well as a matrix, because a virus categorisation can be formed by combining any number of the given characteristics. Helenius writes that the set of characteristics might not be exhaustive and that there might appear previously unknown characteristics in the future. He also points out that a virus always has to have at least one of the two characteristics *memory resident* or *direct action*.

The characteristics in Helenius' scheme [31, pp. 15–17] are:

- polymorphic,
- companion,
- stealth, with subclass tunnelling,
- direct action or memory resident,
- linking, and
- information distributing, with subclasses self-distributing and e-mailing, which in turn have the common subclass self-e-mailing.

4.5.4 Summary of evaluation

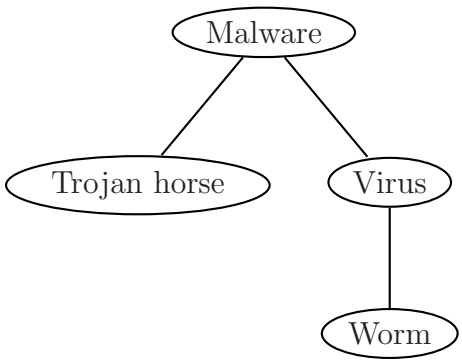
The classification scheme of harmful program code is really not exhaustive, especially not the subclasses of malware, which Helenius also admits. The classification is also somewhat ambiguous, because viruses, worms, and trojan horses are said to eventually include operations typical for other types of malware, but should yet not be classified as such. How to differentiate the malware types in those situations is not specified. Helenius would also need to further explain why the non-harmful (derived from his definition) category *joke programs* is included in the malware class, which he has defined as programs deliberately made harmful.

Regarding the classification scheme based on different infection mechanisms for viruses, it does not specify where to place worms, which are regarded as a subclass of viruses. Consequently a user of the classification scheme needs an implicit understanding of the field to be able to classify a virus or a worm, i.e. the scheme is hard to use in practice.

Also the last presented way of categorising viruses, namely after their (general) characteristics, suffers from not being exhaustive. Moreover, the two categories *stealth* and *linking* are not mutually exclusive, because one way of acquiring stealth is to change the linking of sectors in the file system, which also happens to be the definition of the linking class.

How Helenius relates the three malware types trojan horse, virus and worm to each other is shown in Figure 5.

Figure 5. The relationship of a trojan horse, a virus and a worm according to Helenius.



None of the three categorisation schemes presented by Helenius in [31] does fill all the requirements of a taxonomy stated in Section 3.2.2.1). The parts about viruses are shorter versions of Bontchev's, which was regarded as not detailed enough for filling the needs of FOI (see Section 3.1.3).

4.6 Howard-Longstaff

Howard and Longstaff aim at creating a common language for computer security incidents and therefore also has to categorise the tools used for 'exploiting a computer or network vulnerability' [21, p. 13].

The outline of the proposed incident taxonomy is fairly the same as in [35]. The tool part contains the same categories, but the definitions are

more detailed in the latter. However, that report does not mention anything about the exclusiveness or exhaustiveness of the categorisation. Therefore, only the first one, [21], is evaluated here.

The list of tools used covers a wider spectrum than just software based IT weapons. The software based tools listed are (quoted from [21, pp. 13–14]):

Script or program – a means of exploiting a vulnerability by entering commands to a process through the execution of a file of commands (script) or a program at the process interface. Examples are a shell script to exploit a software bug, a Trojan horse login program, or a password cracking program.

Autonomous agent – a means of exploiting a vulnerability by using a program, or program fragment, which operates independently from the user. Examples are computer viruses or worms.

Toolkit – a software package which contains scripts, programs, or autonomous agents that exploit vulnerabilities. An example is the widely available toolkit called *rootkit*.

Distributed tool – a tool that can be distributed to multiple hosts, which can then be coordinated to anonymously perform an attack on the target host simultaneously after some time delay.

Each category is said to have the possibility to contain any number of the other categories. There is an ordering of the categories from simpler to more sophisticated.[23] When using their taxonomy, often a choice has to be made among several tools. By always categorising by the highest category of tool used, Howard and Longstaff claim the categories become mutually exclusive in practice. Based on their experience they also claim their list of tools is exhaustive.

4.6.1 Summary of evaluation

Howard and Longstaff do not explicitly state how they relate the three malware types trojan horse, virus and worm to each other. Consequently, what is shown in Figure 6 is the relationship extracted from the definitions of their categories, where they use the three malware types as examples.

The software part of their classification scheme is really simple, using only four categories in a strictly hierarchical structure. By specifying that a tool always shall be categorised by the highest category of tool it may belong to, their scheme becomes unambiguous.

[23] Howard and Longstaff do not specify the ordering in more detail, but they give *user command* (not software based and therefore not in the list above, in their list it is written before *script or program*) as the lowest level and *distributed tool* as the highest. Supposedly their list actually is ordered in the same way as it is written.

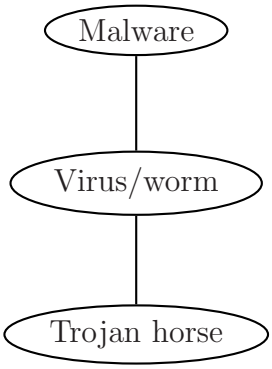


Figure 6. The relationship of a trojan horse, a virus and a worm according to Howard and Longstaff.

However, the category *toolkit* is placed below a distributed tool in the hierarchy of their classification scheme. A toolkit containing (among other things) a distributed denial of service (DDoS) weapon would accordingly be classified as a distributed tool, even if it in practice is even more advanced than such a tool. Thus, their scheme might need an extension and the exhaustiveness may therefore be questioned.

Even if the taxonomy almost (apart from the questioned exhaustiveness) did fill the requirements stated in Section 3.2.2.1 the simple hierarchy with only one alternative in each level is far to coarse to fit the needs of FOI stated in Section 3.1.3 and the taxonomy cannot be used as a taxonomy of software weapons.

4.7 Landwehr

The work by Landwehr et al. outlines a taxonomy of computer program security flaws. They have chosen to use the term *malicious flaw* as a synonym for malware and in that way managed to incorporate the term into their taxonomy.

They acknowledge the difficulties of characterising intention, that it is hard to decide whether a program flaw is made on purpose or not. But they use the term anyway, because as they see it the risk of inadvertently creating a malware is minimal in practice.

A trojan horse is by them specified as [11, p. 6]:

> [...] a program that masquerades as a useful service but exploits rights of the program's user – rights not possessed by the author of the Trojan horse – in a way the user does not intend.

They then define a virus as a trojan horse '[...] replicating itself by copying its code into other program files'. Accordingly a worm becomes a trojan horse that '[...] replicates itself by creating new processes or files to contain its code, instead of modifying existing storage entities'.

They place trapdoors and logic bombs (including time bombs) as separate classes on the same level as trojan horses. However, trapdoors and

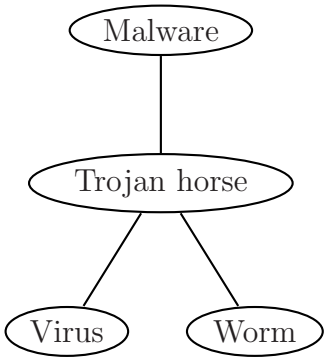
time bombs are said to be possible to include in trojan horses, so the classes are not mutually exclusive.

4.7.1 Summary of evaluation

The uppermost level of the Landwehr et al. proposed classification scheme of malicious flaws is formed by trojan horses, trapdoors, and time bombs or logic bombs. But because of the possibility to incorporate the other two classes into trojan horses, the classes are not mutually exclusive. Consequently the scheme is not detailed enough to fill the needs stated in Section 3.1.4. Nor does the scheme meet the requirements stated in Section 3.2.2.1).

Landwehr et al. regard viruses and worms as subclasses of trojan horses, as shown in Figure 7.

Figure 7. The relationship of a trojan horse, a virus and a worm according to Landwehr et al.



4.8 Conclusion

None of the evaluated taxonomies or categorisation schemes fulfil all the requirements of a proper taxonomy, specified in Section 3.2.2.1. The part of Howard's and Longstaff's incident taxonomy covering software was the one closest to fulfilling the requirements. The reason for this was its simple structure with only one category at each level in the hierarchy. On the other hand, this simplicity made it far from fulfilling the needs presented in Section 3.1.4. Actually, the required level of detail is not available in any of the different categorisation schemes used today.

Regarding the CARO naming convention it should not really have been evaluated as a taxonomy. However, it was included because it is a widely known document, which might have been used to form the basis for a new taxonomy. Unfortunately it was not found to be exhaustive, probably because it is rather old. The new types of viruses missing from the specification were maybe not predicted by the authors. The proposed extension by Scheidl was not detailed enough to bring the naming scheme up to a high enough standard for making the scheme usable as a taxonomy. Also the fact that it is focused on viruses made it too weak on the non-viral side to meet the requirements in Section 3.1.4.

The figures indicating the relationship of the three software weapon types trojan horse, virus, and worm show how differently each classification scheme define these types. Not two figures are alike! If any conclusion is to be drawn from this, there is a tendency of putting all three types on the same level[24], although they in several cases are defined as not being mutually exclusive.

The differences in the figures also clearly show the need for a redefinition of the three terms, a redefinition made from a common base.

[24] Figures 1, 2, 3, and 4 representing $\frac{4}{7}$ of the set

5 TEBIT

The name *TEBIT* is a Swedish acronym for 'Technical characteristics' description model for IT-weapons'[25]. The acronym has only been kept because no better English alternative has been found.

In this section the definition accompanying the taxonomy, as well as the taxonomy will be discussed. The taxonomy has been slightly updated since the publishing of the NordSec paper. Therefore all the categories have got their own subsection containing the motivation for including them, the changes made, as well as any other necessary information.

Note that the information presented in the NordSec paper will not be repeated (not on purpose anyway). The reader is therefore recommended to read the paper (see Appendix A or Section 2.2) before reading this section.

5.1 Definition

The definition used is based solely on measurable characteristics of software weapons. As stated earlier in the text, the definition reads as follows:

> A software weapon is software containing *instructions* that are necessary and sufficient for a *successful attack* on a *computer system*.

The italicised words are all explained in Section 2.1.2, paragraph *New Definition* and then the first three words are further explained in the subsections below.

5.1.1 Instructions

Because the *instructions* constituting the code are to be used to decide whether a tool is a software weapon or not, they have to be available in a readable format. How this is to be achieved falls outside the scope of this thesis, but a few possible ways might be mentioned anyway.

First of all the compiled code may be possible to decompile or in any other way reverse-engineer. For example the anti-virus companies are sometimes using such methods when dissecting new strains of computer viruses. Professor Klaus Brunnstein, head of the Virus Test Center (VTC) in the Computer Science Department at the University of Hamburg, Germany has been teaching reverse engineering methods to students since 1988 [43]. Thus the methods are there and possible to use.

Secondly, tools simulating complete virtual network environments exist. These can then be used to study the behaviour of different software weapons and in that way give a rather good idea of the technical characteristics of the weapons. One problem with this method is that the exhaustiveness of the study is undecidable, there is no way of proving that all

[25] 'Teknisk beskrivningsmodell för IT-vapen' in Swedish.

properties have been found. Not even a lower boundary of the quality of the study is possible to calculate. The problem might be compared to software testing (debugging) and quality control, but in the software weapon case there is often no specification available to tell what the software is expected to do.

There is always the possibility that the source code of a weapon might be available in some way. Then the only thing required is programming skills in the language used and such a thing is always achievable.

5.1.2 Successful

For a software tool to be a weapon there has to be at least one system (real or modelled) containing a vulnerability or exposure that the software tool uses to violate the computer security of the system. The vulnerability or exposure does not have to be known in advance, as soon as a software tool violates the computer security in any way, it is to be regarded as a software weapon. Nor has the system to be on the market or in a working condition. It is enough that the weapon violates the computer security of a system in development, or simply any algorithm that might be included in a future system, because if the system or algorithm is ever used, it will be vulnerable to that specific weapon.

This was not clearly stated in the NordSec paper (see Section 2.1.2.2, *Successful*). The text somewhat contradicted itself, because it was first stated that at least one system had to be vulnerable, then that a used vulnerability did not have to be part of an existing system. As stated above, it is enough to have proven that the weapon will break the security of a system *as soon as* that system exists.

5.1.3 Attack

Regarding the definition of *attack*, some terms can be further explained. First of all the definition of computer security is not generally agreed upon. [10, 49, 50] However, the inclusion of the three objectives *confidentiality*, *integrity*, and *availability* is almost unanimous.

In the NordSec paper the definitions of the terms where cited from Gollmann [10, p. 5] (who cited ITSEC). A fairly similar definition is the following one, quoted from Common Criteria (CC) [51, p. 14]:

> Security specific impairment commonly includes, but is not limited to, damaging disclosure of the asset to unauthorised recipients (loss of confidentiality), damage to the asset through unauthorised modification (loss of integrity), or unauthorised deprivation of access to the asset (loss of availability).

The real difference is that CC uses the word ‘damaging’ and ‘damage’ in the definitions of confidentiality and integrity, which Gollmann and ITSEC

does not. As seen above even CC acknowledges the core as these three terms and agrees that sometimes also other terms are included.

In [24, p. 6] *vulnerability* (and the accompanying term *security policy*) is defined in the following way, which is quoted verbatim from the source:

> **Vulnerability** is a condition in a system, or in the procedures affecting the operation of the system, that makes it possible to to [sic!] perform an operation that violates the explicit or implicit security policy of the system.
>
> **Security policy** is some statement about what kind of events are allowed or not allowed in the system. An explicit policy consists of rules that are documented (but not necessarily correctly enforced), while an implicit policy encompasses the undocumented and assumed rules which exist for many systems.

Another definition is the one used in [12]. It is a long text, but the main idea is that the term *vulnerability* can have two different interpretations, one wide and one narrow. In the first case a vulnerability is regarded as a breaking of the security of a computer system in some context. The more narrow interpretation concerns only deviations from the specification of the functionality of a software, somewhat in accordance with Brunnstein's definition of dysfunctional software (see Section 4.3). In this way programs that work as specified, but in an insecure way, are not regarded as containing any vulnerabilities.

Because there are several interpretations of vulnerability, the Common Vulnerabilities and Exposures (CVE) Editorial Board decided to use the term *exposure* to work together with the narrow interpretation, in order to make the two alternative definitions more equal. The term exposure is then defined as everything relating to computer security not regarded as being a vulnerability by some, but still introducing a weakness into the affected program or computer system. The definition proposed by the CVE Editorial Board is not a strict one and is expected to change over time. However, in 1999 they voted to accept a *content decision* describing the terminology to be used in CVE, ratifying the proposed definitions discussed above. [12]

Because the definition and the taxonomy are created from a technical point of view, the preferred definition of a vulnerability (and exposure) is the CVE one. Also the definition made by Lindqvist (see above or [24, p. 6]) is applicable, with the restriction that it should only be weaknesses in the software constituting the system, that are used to violate the security of the system. In other words, a bad security policy or careless users are not to be regarded as vulnerabilities (even if they in some cases really might be qualifying as such . . .).

5.2 Taxonomy

A few changes have been made to the taxonomy since the publication of the NordSec paper. Each change is discussed in depth in Section 5.3. The original definitions of the categories are shown in Section 2.2.

The current taxonomy consists of 15 categories (see Table 2), but new changes might be needed after the taxonomy has been further tested.

Table 2. The taxonomic categories and their alternatives, updated since the publication of the NordSec paper

Category	Alt. 1	Alt. 2	Alt. 3	Alt. 4
Type	atomic	combined		
Violates	confidentiality	integrity; parasitic	integrity; non-parasitic	availability
Duration of effect	temporary	permanent		
Targeting	manual	autonomous		
Attack	immediate	conditional		
Functional area	local	remote		
Affected data	stationary	in transfer		
Used vulnerability	CVE/CAN	other vuln.	none	
Topology of source	single	distributed		
Target of attack	single	multiple		
Platf. depend.	dependent	independent		
Sign. of repl. code	monomorphic	polymorphic	not repl.	
Sign. of attack	monomorphic	polymorphic		
Sign. w. passive	visible	stealth		
Sign. w. active	visible	stealth		

All categories in the taxonomy are independent, but they are not mutually exclusive. If they were, the taxonomy would not be possible to use, the requirement to use at least one alternative from each category would contradict the mutual exclusiveness. As soon as one category was used, the others would be disqualified.

Instead the alternatives in each category, except for the category *Violates*, are mutually exclusive and unambiguous (based on an empirical evaluation). Together the alternatives in a category form a partitioning of the category and thus they also are exhaustive. The alternatives in the category *Violates* are only disjunct, not mutually exclusive.

Another way of describing the taxonomy is to view it as a 15-dimensional basis spanning the set of all software weapons. Then the mutual exclusiveness, exhaustiveness, and unambiguity come naturally.

Regarding the usability the names of the categories and alternatives are selected to be as short and at the same time as descriptive as possible. The chosen level of abstraction of the properties described by the categories are rather high to make the taxonomy future proof, yet detailed enough to satisfy the requirements of FOI regarding the scenario development tool. The number of possible categories are

$$2 * (2^2 - 1)^{11} * (2^3 - 1)^2 * (2^4 - 1) = 260406090$$

which is more than enough to fulfil the needs of FOI.

The format of the taxonomy can be either a (two-dimensional) matrix, or a one-dimensional vector. The real difference is that the matrix is easier to read and understand, furthermore it is compact. The vector format can be viewed as a 34-bit binary string which facilitates the comparison of different categorisations. The definitions of the categories and alternatives are the same for the two views, they are just different ways to view the same thing. This is further discussed in Section 5.3.1.

5.3 In depth

In this section each category of the taxonomy is discussed regarding its interpretation, why it should be included in the taxonomy (its relevance regarding the needs stated in Section 3.1.4), and a brief discussion of possible countermeasures.

The explanations of the different categories presented in the NordSec paper are repeated in Section 2.2. In this section changes made since the publication of the paper and further explanations will be given. The user is therefore recommended to read Section 2.2, which explains the background of the categories, before reading the text in the following subsections.

5.3.1 Type

Many of the software weapons created today are complex combinations of simpler, already existing software weapons. Consequently, there is a need for a possibility to separate the simpler weapons from the combined ones. Therefore this category has to be included in the taxonomy.

An atom is the smallest part of a software fitting the definition of a software weapon. There is actually no guarantee that an atom is not using more than one alternative from each category, which then might contradict the mutual exclusiveness of the alternatives.

One example of this is the *Affects* category. Think of the simplest form of a parasitic virus; the only thing it does is replicate and create new copies of itself within the host file. Of course this affects the *integrity; parasitic* of the system, but because of the lack of a birth control function, it will also eventually fill the hard disk and maybe crash the system. Thus it also affects the *availability* of the system.

An alternative to the definition of *Type* used in the NordSec paper (see Section 2.2.1) is to regard the category as indicating if there is more than one alternative used in any category. The alternative *atomic* would then be changed to the more appropriate *single*. Such a weapon would use exactly one alternative from each category and the violation of the mutual exclusiveness would be avoided. However, if that solution is used, there is no way of telling the different atoms forming a combined weapon apart.

Nor is it possible to learn from the categorisation of a weapon, whether it is an atom or not, and that is a high price to pay.

However, if the matrix form of the taxonomy is abandoned in favour of a one-dimensional form, the problem of having to use several mutually exclusive alternatives together might be solved. The taxonomy then can be seen as 34 new categories constructed from the combination of an old category and an alternative from that category. These new categories can be either *true* or *false* (or 1 or 0). But then the taxonomy will be rather hard to take in and for an atomic (not nuclear!) weapon approximately half of the categories might be redundant. Worth noticing is that the requirement of using at least one alternative from each old category is not relaxed. One advantage of the one-dimensional form is that it will make the checking of the deviation between the categorisations of two weapons easier.

The chosen alternative is to retain the matrix form and use it where suitable, because it increases the readability. However, the preferred definition of the category *Type* is the latter one indicating the categorised weapon being an atom, and with no connection to how many alternatives that may be used in each category. But, in most cases dealing with an atom, there probably will be only one applicable alternative in each category, except maybe for the category *Violates*.

One important thing to observe is that in the category *Type* it is *not* possible to combine the two alternatives atomic and combined. Either the categorised weapon is an atom, or it is not. In other words this category is really a meta-category, one level above the other fourteen.

5.3.2 Violates

For a software tool to be defined as a weapon, it has to violate at least one of the fundamental properties of computer security, namely *confidentiality*, *integrity*, or *availability*. But these alternatives are actually possible to combine, i.e. they are not mutually exclusive, only disjunct. Despite of this fact the category, which is renamed from *Affects* to *Violates*, is fundamental for the categorisation of a software weapon and thus a natural part of the taxonomy.

However, more than the name of the category has changed since the publication of the NordSec paper (see Section 2.2.2). The alternative *integrity* has been divided into two parts; *integrity; parasitic* and *integrity; non-parasitic*. The new alternatives are meant to provide a way to separate software weapons needing a host to hold their code (*integrity; parasitic*) from weapons capable of sustaining their code themselves (*integrity; non-parasitic*). The alternative parasitic is not restricted to viruses only, it also includes weapons for instance needing to piggy back on another file to be able to transfer themselves to new hosts.

A parasite[26] in some way connects its own code to code found in the

[26]The term is defined in the following way in [33]: '[BIOLOGY] An organism that lives in or on another organism of different species from which it derives nutrients and

attacked system, i.e. a host file. The parasite does not replace its host, it needs part of the host to be alive[27] to function. Thus, a file virus which completely replaces all of the host file content, only leaving the linking of the host file in the file allocation table (FAT) untouched, is not parasitic. This is also true if the virus keeps the two first bytes in the `.EXE` file it infects. The bytes are the same for all `.EXE` files and are not to be regarded as a unique part of the code.

Nor is a boot sector virus a parasitic virus, at least not if it replaces all the original code in the boot sector with its own code. The only thing remaining is the dead shell, the physical or virtual position, of the original code. Compare the situation to that of the non-parasitic hermit crab, which uses the abandoned shells of gastropods as a protective cover [52].

However, if the boot sector virus does not retain the original content of the boot sector somewhere, the system will crash immediately after the execution of the virus code. The virus does replicate, but the chance of noticing such a poorly implemented virus is big.

These two new alternatives of the category are needed in the taxonomy to make a categorisation more exact regarding how a software weapon is sustaining itself, if it needs a host file or not to function.

The alternative solution of having a new category called *Subsistence* added to the taxonomy is not preferable, because the alternative *integrity* would be dependent on the alternative *parasitic* in the new category.

There are no 100% good countermeasures against weapons violating either of the four alternatives in the category. The possible countermeasures to be used against respective alternative are:

Confidentiality violating weapons may be counteracted by using cryptographic methods to render the information unreadable. These methods can also be used to authenticate the reader of the information.

Integrity; parasitic may be enforced by write-protecting files as far as possible. Some kind of hashing (checksumming) of the executable files in the system might also help.

Integrity; non-parasitic In this case some type of cryptography may be used to sign and verify the correctness of data.

Availability is hard to uphold, but by identifying possible bottlenecks and then building some redundancy into the weak spots the resistance against attacks on the availability can be improved.

5.3.3 Duration of effect

This category is included in the taxonomy, because it is vital to know if a weapon does permanent damage to an attacked system or not. Of course

shelter.'

[27] Here defined as needing some unique part of the code to be present.

a violation of the computer security of a system is serious no matter what, but when dealing with for instance risk assessment and grading of the seriousness of the threat from a specific weapon, the duration of the effect is good to know.

Sometimes a conventional weapon is defined as being a tool used to inflict irreversible damage. The same definition is not fully applicable in the world of computers, where it is possible to make identical copies of files, which then can be used to restore a destroyed system. However, using such an approach most attacks using software weapons may be regarded as doing only temporary damage. Therefore the definition of this category does not take the possible backup and restoration facilities into account. It is the duration of the effect of a weapon on a system where no restoring operations are performed that should be measured.

The characteristic of a software weapon described by this category is too general to have any specifically applicable countermeasures. What can be done is taking regular and frequent backups of the data and also always install new software patches as soon as they are published.

5.3.4 Targeting

Both a manually aimed weapon and an autonomously targeting weapon can do severe damage. Often it is harder to defend a system against a manually targeted weapon, because it can be more accurately aimed. In that case there is also most probably a human being supervising the execution of the weapon and that is the most formidable enemy to face. Of course the address of the attacked system can be changed, but then no other parties will be able to reach the victim either.

But on the other hand an autonomously targeting weapon can infect a large amount of computers in a very short time. It is also possible to think of the use of artificial intelligence to build agents that choose their targets in a more systematic way than simply using random IP addresses, as often is the case today.

A possible countermeasure is changing the address of the affected system when it is being attacked, but that will then turn into a very effective attack on the availability of the system instead. Other thinkable (and less radical) methods is to use a firewall or network address translation (NAT) server to hide the actual addresses of the system.

5.3.5 Attack

The concept of flow control is a cornerstone in programming and thus many weapons use conditions to govern their attack pattern. These weapons often are called logic bombs, mines etc. Also other types of weapons use conditions to vary their behaviour. Therefore this category is an essential part of the taxonomy.

In this case the general countermeasure is to use a sandbox[28] to run suspicious code in. Also the perimeter defences must make sure that no software weapons are let into the system.

5.3.6 Functional area

The idea is to separate a weapon using some sort of remote procedure call (RPC), remote method invocation (RMI), or the like to execute an attack, from a weapon attacking the computer it is residing on. Also other ways of attacking a remote computer is thinkable. A weapon not residing on the attacked computer may be harder to defend against, it might also be harder to detect, at least in its passive phase. Consequently, to be able to take the correct defensive actions this category is of utter importance.

To defend a system against weapons of this kind the system has to be properly updated with all the latest software patches. No unnecessary[29] rights should be given to code in the system and the code has to be thoroughly debugged and tested. Also the perimeter defences (firewalls, NATs, etc.) have to be in place and properly configured.

5.3.7 Affected data

Especially weapons affecting data in transfer can be effective in preparing or even execute an attack. By sniffing the network traffic valuable information can be retrieved. Both the general topology of the network and more specific information, such as passwords and cryptographic keys, are valuable for making a successful attack possible. Also for example a man-in-the-middle weapon can be used. In that case the attacker more or less controls every single bit sent between two users. Therefore this category needs to be included in the taxonomy.

This category indicates if the attacked data (everything not being hardware) is stationary or transported over a network. Its name has been changed from *Sphere of operation* to *Affected data* to better indicate this. Also the alternatives of the category are renamed from the version presented in the NordSec paper (see Section 2.2.7). However, the underlying idea is not changed.

Regarding the characteristic represented by this category the use of good cryptographic solutions will strengthen the defences by making the data unreadable by an assailant. The communicating parties need to use cryptographic authentication methods when initiating a session. The rights

[28] A virtual area in the computer isolated from the rest of the system, where untrusted code can be executed without affecting the rest of the system.

[29] Sometimes programs need to have access to services usually reserved for the core of the operating system. Occasionally a programmer gives such rights to the program as a whole, even if it is not needed. These rights might then be violated by utilising a software vulnerability in the program.

of the different involved parties also have to be thought through. If possible the right to write to files should be restricted.

5.3.8 Used vulnerability

By knowing the exact vulnerability used by a weapon to violate the security of a computer system, it is possible to directly see which software patches protect a system from a specific weapon. Therefore this category is meant to be used together with for example the ICAT [17] and CVE [13] databases to facilitate such a function.

The alternative *other method* in the category is changed to *other vulnerability*. In this way the alternative better reflects the fact that some sort of vulnerability is used, even if it lacks a CVE/CAN name. To make the alternatives of the category exhaustive, the alternative should be defined as all technical (not user related) vulnerabilities and exposures not listed by CVE. The use of the alternative *none* then will indicate that the weapon needs help from a user in the attacked system to be able to execute.

In this way a trojanic property can be indicated by defining a trojan horse as a piece of software which is not using any software vulnerability or exposure to violate the security of the attacked system. In other words, it dupes a user to execute its code to be able to perform the intrusion or attack.

The preferred definition of a vulnerability or exposure is the one used in connection to the CVE meta-base (see Section 5.1 or [12]).

The best way to keep the defences high is to install all the latest software patches as soon as possible. Another important thing is to implement a good security policy, which has to be followed by *every* user in the system.

5.3.9 Topology of source

A massive, synchronised attack from several concurrent sources is really hard to find countermeasures for. By distributing daemons to a large number of computers and then activate them by remote control, a synergy effect can be attained, because a larger (virtual) area of the Internet is affected. An attacker grossly outnumbering the victim of an attack has a huge advantage, and if also the sources of the attack can be spoofed, the victim is more or less defenceless. Thus, the inclusion of this category in the taxonomy is very important.

Because the distributed daemons are alike, they also are affected by the same type of countermeasures. If it is possible to identify the software weapon used for the attack, there might be ways to shut the daemons down by sending special instructions to them. Then all of them might shut down at the same time. Of course this is only possible if the source addresses are authentic.

Also the system needs to be well patched and using good perimeter defences.

5.3.10 Target of attack

Some weapons have the ability to concurrently attack several targets at once, for instance to make the spreading of the weapon more effective. One example is CodeRed [53, 54, 55] using 100 threads at the same time to search for new victims to infect. Also other types of weapons are thinkable, a tool searching for vulnerabilities scanning a whole range of IP addresses at once, for example. The multi-threaded weapons can spread very quickly and therefore amount a considerable threat to computer systems. Thus the category has to be included in the taxonomy.

To be able to defend against software weapons with the characteristics described by this category, the use of an IDS is recommended. By checking the behaviour of the code executed in the system and taking action when something unusual is discovered (for example generating a lot of threads), the system might have some protection, or at least other systems might be somewhat protected from attacks emanating from this system.

5.3.11 Platform dependency

Weapons able to run on several different platforms of course have a greater ability to do widespread damage and may therefore be more dangerous than single platform weapons. Also the increasing use of mobile code and interactive web sites for commercial purposes have added to the vulnerability of the modern society. Therefore it is important to be able to correctly categorise a software weapon with the ability to run on several platforms and hence the category has to be part of the taxonomy.

There are no real platform independent programming languages or softwares yet, at least not in a strict sense. Therefore the choice to use the word *independent* as the name of the second alternative might be disputed. What the alternative is meant to indicate is that the weapon is not depending on one single platform to run, instead it has the ability to run on several platforms. The word independent is logically the complement to dependent. It is also a way of making the category future proof. In a few years time there might exist real platform independent programming languages and thus also real platform independent software weapons.

All computer programs need a processor to run on, many of them also need one or more operating system application program interfaces (APIs) (or other operating system specific modules) to be able to execute. The most important API in an operating system needed by an executable file should be chosen to represent the operating system. One example of such an API is Win32 in the Microsoft Windows series of operating systems.

To properly include languages running on any kind of emulators the word *processor* should not be literally interpreted. Take the Java language

as an example. When compiling a Java program it is converted into byte code. The compiled program then can be executed on a Java Virtual Machine (JVM), which is available for many different combinations of operating system and processors. What the JVM really does is converting the byte code into machine code specific to that particular combination of processor and operating system. Therefore the JVM can be likened to a processor from the Java program's point of view. Actually there also exist several different Java hardware processors [56, 57, 58].

The same thing is true for macro or script languages. For example the Visual Basic for Applications (VBA) scripts used in Microsoft Office as macros are using the VBA interpreter as a processor to run on, irrespective of the operating system installed on the computer. Therefore they are to be regarded as platform independent.

From the code of a program it is possible to see whether the program uses any APIs or not. It is also possible to see if the program needs a specific processor to run on. If this is abstracted to a two dimensional graph with the processor type as the x-axis and the needed operating system specific APIs as the y-axis, it is possible to describe the platform dependency graphically. This is done through counting how many points the particular program is represented by in the graph.

A program needing an API specific for one operating system and to be run on one specific type of processor is represented in the graph as one single point, which makes it a platform dependent program (see Figure 8). The program will still be platform dependent even if there exists a software emulator implemented in another operating system or on another type of processor than the program can run on, because the code of the program still indicates that it needs both a specific API and a specific processor to execute.

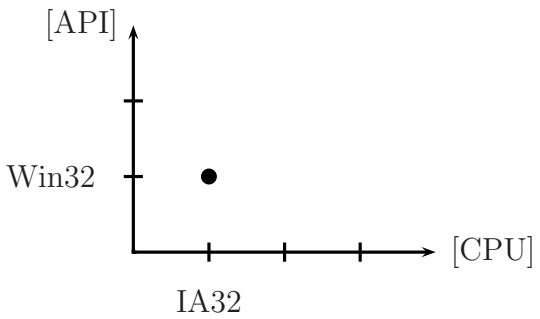
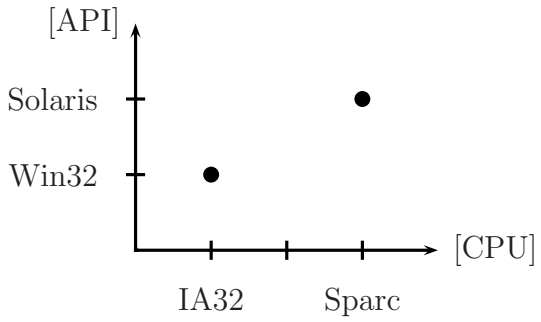


Figure 8. This is an example of what the graph of a platform dependent program would look like.

If the program can use APIs from different operating systems and run on one or more type(s) of processor(s), or vice versa, it is represented by two or more points, and thus is platform independent (see Figure 9). There are examples of software weapons containing code written in two different languages, for example the proof-of-concept virus Peelf [59].

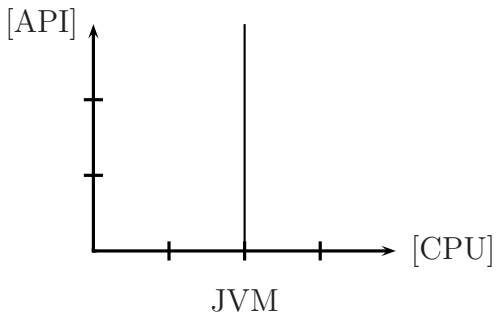
A program not needing any specific APIs to run will be represented by one (or more) vertical line(s), i.e. be platform independent (see Figure 10). This can for example be a Java program, or a Microsoft Office macro. Also

Figure 9. This fictive platform independent program is represented by two points.



a program written purely in machine code and not needing any operating system to start would be platform independent. One thinkable example is a software weapon which is active before the operating system is loaded. This might seem to be wrong, because the software weapon might only be able to run on a very narrow series of processors. However, it would be able to run regardless of the type of operating system installed on the computer.

Figure 10. The platform independent program not using any particular API is represented by a vertical line.



The platform dependency category might seem to suffer from the same problem as the malware definition, it is the context of the program that decides whether a program is platform dependent or not. When new operating systems are developed they in most cases are made compatible with older versions of operating systems from that manufacturer. They might also contain APIs making them compatible with operating systems from other manufacturers.

If the platform dependency is to be measured by the number of operating systems able to run the program, and not the requirements of the program itself stipulated in its code, almost all programs would sooner or later become platform independent. This would not be a good solution and hence the platform dependency is to be decided solely from the code of the program. In that way the categorisation of a software weapon would not have to be changed if a new all-in-one operating system was developed.

The platform independent software weapons (as well as the dependent ones) can be counteracted by running all untrusted code in a sandbox. The policy to only use code verified to be good by a trusted party may also help.

5.3.12 Signature of replicated code

If not the look, the signature, of the thing being searched for is known, it is much harder to find. Thus, a replicating weapon with the ability to vary the signature of its code is of course much harder to detect by scanning for code signatures, than a weapon not changing its signature. Consequently, this characteristic is of great importance, because it makes it possible to distinguish polymorphic weapons from monomorphic or not replicating weapons. As a bonus it also gives an opportunity to indicate whether a weapon replicates or not.

Therefore the name of the category was changed from *Signature of code* to *Signature of replicated code* and the alternative *not replicating* was added. Another reason for the change was to be able to separate non-replicating weapons and monomorphic weapons, which got lumped together when using the old version of the category.

The use of heuristic scanners as well as behaviour checking tools will decrease the risk of being successfully attacked by polymorphic weapons. Regarding the monomorphic weapons they are possible to defend against using signature scanners, which is also applicable to non-replicating software weapons.

5.3.13 Signature of attack

An important property of a software weapon to know about is the ability to adapt to the defensive strategies used by its victims, because such a weapon is much harder to find countermeasures for. Thus, this property must be included in the taxonomy.

The definition of this category used in the NordSec paper was formally correct, but not really useful. It restricted the alternative *polymorphic* to weapons able to vary their attack signature between attacks of the *same* type.

First of all, by using that definition the category becomes ambiguous. It is left to the reader to decide whether an attack is of the same type as another. Nothing is mentioned of the level to make the distinction on, if it is defined as using the same vulnerability, or if it is some kind of meta level attack, as for example a denial of service attack.

Secondly, if all users of the taxonomy managed to interpret the category in the same way, there was a risk of imbalance in the usage of the two alternatives. Depending on the interpretation used, either almost no weapons (of those existing today) would be placed in the polymorphic group (when using the stricter interpretation), or the situation could be reversed (when using a meta-level interpretation).

Therefore the definition of the category is changed to distinguish between weapons able to in some way vary their method of attack independently, without user intervention (*polymorphic*), and those using a preset attack method (*monomorphic*). The variation can be random or governed by

rules, the important thing is that the weapon is changing the attack type on its own. When a rule set is used the weapon can adapt its behaviour to the defensive actions taken by the victim.

An example of a weapon using a random attack pattern is the Tribe Flood Network 2000 (TFN2K) distributed denial of service weapon which can randomly switch between four different attacks (see Appendix B and [60]).

An IDS may somewhat protect from attacks performed by software weapons having the ability to polymorphically change their attack pattern, by detecting them and maybe even adapt to their behaviour.

5.3.14 Signature when passive

By using some kind of stealth technique a software weapon can hide its presence from defensive softwares. When the weapon is not performing any attack related operations, it is in a passive phase. If stealth is used the software weapon can lie hidden in a system for a long period of time and wait for the right moment to attack. Therefore it is important to know if a software weapon has such a characteristic and the category consequently needs to be included in the taxonomy.

There are no really good countermeasures against weapons using stealth techniques, more than never letting them seize the opportunity to attack. In other words they have to be stopped at the gates (by a firewall, NAT, or the like).

5.3.15 Signature when active

As stated in the *Signature when passive* category (Section 5.3.14) it is important to know if a weapon uses stealth techniques to hide. Such techniques can also be used by the weapon when it is in an active phase, i.e. is performing operations representing an attack on a computer system. Consequently this category is equally important and thus needs to be included in the taxonomy for the same reasons.

The countermeasures available to defend against weapons using stealth techniques for hiding their active phases are the same as those used against weapons using stealth during their passive phases (see Section 5.3.14).

5.4 In practice

As stated in Appendix A the proposed taxonomy needs to be tested. A proper testing of its usability would require several different persons to independently classify a set of software weapons. The resulting classifications would then need to be completely similar to indicate that the taxonomy might be usable in practice (after corrections of the direct misunderstandings during the classifier's learning phase). Also the testers gen-

eral opinions on the usability of the taxonomy would have to be collected and any complaint or remark be properly attended to.

To facilitate such a test (at least a small one) nine software weapons have been classified by the author. The result is shown in Appendix B. The reader of this thesis then may test the taxonomy on his or her own by classifying these weapons. The weapons and the references used for the classification are shown in Table 3.

Table 3. The categorised weapons and the references used for the categorisation

Software weapon	Reference(s)
mstream	[61]
Stacheldraht	[62, 63]
TFN	[64]
TFN2K	[60]
Trinoo	[63]
CodeRed	[53, 54, 55]
CodeRed II	[65, 66, 67]
Nimda	[68, 69, 70]
Sircam	[71, 72, 73]

For the taxonomy to be usable in practice it has to classify phylogenetically[30] related software weapons in a fairly similar way. Of course weapons not related to each other then would have to have deviating classifications.

The best way of measuring the level of diversity in a group of software weapons can be discussed. The method chosen in this thesis is to represent each alternative as a one-dimensional 34 bit long binary string, which is mathematically represented as a column vector s (see tables in Appendix B). Each bit of the string has the value given by s_i where $s_i \in \{0, 1\}$ and $i = 1, 2, \ldots, 34$.

A collection of $m \geq 1$ categorised software weapons, all represented as vectors in accordance with the format described above, together form a set T. This set then can be described as a matrix where s_{ij} is the value of the i:th bit in the categorisation of software weapon j.

To measure the level of difference the standard deviation, here called d, is calculated for each row i in T. Thus the formula becomes

$$d_i = \sqrt{\frac{1}{m-1}\sum_{j=1}^{m}(s_{ij} - \overline{s_i})^2}$$

The nine weapons were divided into two sets, T_{DDoS} and T_{worms}. The resulting standard deviation d_i for each set is shown together with the d_i for the whole group (T_{all}) in Table 4. The last column shows where $d_i = 0$ in each group, but $d_i > 0$ for the complete set. In other words it indicates

[30] In [33] phylogeny is defined in the following way: '[EVOLUTION] The evolutionary or ancestral history of organisms.'

which combinations of category and alternative that may be used to distinguish the two groups from each other. If $d_i > 0$ in the *Disting.* column in the table, that specific combination of category and alternative distinguishes the two groups.

Table 4. The standard deviation d_i of T_{DDoS}, T_{worms}, T_{all}, and the distinguishing alternatives ($d_i > 0$)

Category	Alternative	T_{DDoS}	T_{worms}	T_{all}	Disting.
Type	atomic	0	0	0	0
Type	combined	0	0	0	0
Violates	confidentiality	0	0.5	0.5	0
Violates	integrity; parasitic	0	0.5	0.5	0
Violates	integrity; non-parasitic	0	0.5	0.5	0
Violates	availability	0	0.58	0.44	0
Dur. of effect	temporary	0	0.58	0.44	0
Dur. of effect	permanent	0	0.5	0.5	0
Targeting	manual	0	0	0.53	0.53
Targeting	autonomous	0	0	0.53	0.53
Attack	immediate	0.45	0.5	0.44	0
Attack	conditional	0.45	0.5	0.53	0
Funct. area	local	0	0	0.53	0.53
Funct. area	remote	0	0.5	0.33	0
Affected data	stationary	0	0	0.53	0.53
Affected data	in transfer	0	0.5	0.5	0
Used vuln.	CVE/CAN	0	0.5	0.5	0
Used vuln.	other vuln.	0	0	0	0
Used vuln.	none	0	0.5	0.5	0
Topol. of source	single	0	0	0.53	0.53
Topol. of source	distributed	0	0	0.53	0.53
Target of attack	single	0	0.5	0.5	0
Target of attack	multiple	0	0.58	0.44	0
Platform depend.	dependent	0	0	0	0
Platform depend.	independent	0	0	0	0
Sign. of repl. code	monomorphic	0	0	0.53	0.53
Sign. of repl. code	polymorphic	0	0	0	0
Sign. of repl. code	not replicating	0	0	0.53	0.53
Sign. of attack	monomorphic	0.45	0	0.33	0
Sign. of attack	polymorphic	0.45	0	0.33	0
Sign. when passive	visible	0	0.58	0.44	0
Sign. when passive	stealth	0	0.58	0.44	0
Sign. when active	visible	0	0.58	0.44	0
Sign. when active	stealth	0.55	0.58	0.53	0

The result of the standard deviation calculations indicates that the T_{DDoS} was more homogeneous than T_{worms}. No further conclusions will be drawn from this than that the term *worm* is more general than DDoS, but that was no big surprise. As seen in the table there is a possibility to differentiate between members of the two groups by looking at how the following categories and alternatives are used:

- *Targeting*

- *Functional area*; local
- *Affected data*; stationary
- *Topology of source*
- *Signature of code*; monomorphic
- *Signature of code*; not replicating

However, the statistical selection is not very large and the results should therefore not be taken too seriously. The two categories *Targeting* and *Topology of source* empirically seem good to use, but maybe the preferred method of differentiating classes should be to theoretically find the most prominent properties of each class not belonging to both of them.

As shown in Sections 4.1– 4.7 the definition of the term *trojan horse* differs among many of the computer security experts and the same is true for such terms as *virus* and *worm*. Also the way the three terms are related to each other differs, which is shown in Figures 1– 7. These things indicate a problem, because when the researchers cannot agree on as fundamental terms as these, they certainly cannot agree on others. One way of solving this dilemma is to make a fresh start, by defining new terms, or redefining already existing terms and their relationships.

To do this the one-dimensional format of the taxonomy is used. Each combination of category and alternative considered relevant for the term in question is marked with either a 1 or a 0, and the other combinations are marked with wildcards.

By comparing the number of (and which) categories that are relevant (not having wildcards) for a certain term, the relationship of the different terms can be established. The computer security field of research will in that way get a new and generally agreed upon standard of terms. Then all efforts can be put into finding countermeasures to the ever increasing amount of software weapons being created.

In Appendix C a proposed way of defining the three terms *trojan horse*, *virus*, and *worm* is given. These categorisations are to be regarded as recommendations. The computer security research community has to agree on the proper definitions together. By using this taxonomy as a base, that mission will be lot easier to accomplish.

6 Discussion

What everything in this thesis really centres around is computer security and therefore also how to secure computer systems from attacks made with the help of software weapons. Now the issue of how to defend a computer system against software weapons is big enough to fill many a PhD thesis. Therefore it will be dealt with only briefly in one part of this section. After all, the thesis is meant to be a master's thesis, nothing more.

Before the summarising discussion ending the thesis, the future issues regarding the proposed taxonomy and its further development will be analysed in a separate part.

6.1 General defences

There is an enormous amount of books, papers, articles, reports, etc. published on how to secure a computer system against intrusion or other non-authorised activities. This section therefore will only shortly introduce the basic ideas of how to minimise the risk of a successful attack involving the use of a software weapon.

The protective measures taken to defend a computer system can be divided into three parts; *prevention*, *detection*, and *reaction*. [10, pp. 4–5] Other names and ways to divide them have also been used, in [74] they instead are referred to as a dichotomy[31] (see Figure 11).

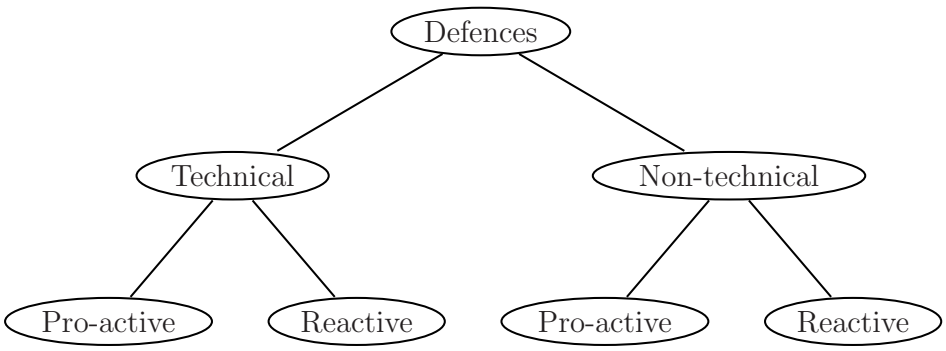


Figure 11. The dichotomy of defensive measures presented by Roebuck. [74]

Irrespective of which scheme is used, the idea is to avoid as many as possible of the dangers by being prepared, both by being precautious and by using technical aids. If anything happens, and it always will, there also have to be (working) methods to decrease the damages and restore the system to normal operation again.

It is easy to draw a parallel to fire fighting. Certainly most people are aware that nothing is really fire proof, also a brick house will burn if drenched in petrol. Also there are no fail-safe technical countermeasures, sprinkler systems need water and maintenance to function and even if fire retardant materials burn less intensely, they still burn. And if there is a fire,

[31] From the Greek word *dihkotomia*, a cutting in two. [32]

the faster and more well organised the fire-fighting, the less devastating the result. Finally, one single fire is all it takes to suffer damage. [74]

Unfortunately, in computer security the focus is on (imperfect) technical countermeasures. The pro-activity and strong reactivity are almost forgotten [74]. Most of the efforts are put into developing better anti-virus scanners, firewalls and IDSs. These efforts are of course not wasted in any way, the methods are needed as perimeter defences and for detection of unauthorised behaviour, but if anyone finds a new ingenious way to do harm, they are of little or no help. Furthermore the methods are reactive, not pro-active, and thus always struggling to keep pace with the development of new software weapons. Also the security in operating systems, the core of all computer systems, is concentrated to stopping unauthorised users from entering the system, while those being let in sometimes get almost unlimited rights.

Therefore the security policy used at the site is of utter importance. When a good policy is adhered to, it functions as a pro-active defence. To be good it has to be strict, but yet allow the users to do their job. If too high a level is enforced the users feel the system is working against them and they tend to invent short-cuts to circumvent the implemented security measures.

The policy has to be supported by the management, too. After all, they decide how much money there is to pay for the computer security. Hence, if they think that enforcing security is a way to waste money, instead of a way to prevent money from being lost, there is an uphill battle to fight for the IT staff at the site.

One very important aspect of the computer security policy is to include continuous education of both the users, the IT staff, and the managers. They all need to be aware (to different degrees) of the threats posed at the moment. Also they need to be constantly reminded of the importance of following the computer security policy.

Regarding the technical aspects of the defence there have to be some sort of anti-virus scanner installed on each and every computer. Preferably there should be scanners from different suppliers installed on the servers versus the workstations or laptops. In that way the risk of a new virus not being included in an update of the signature database is decreased.

This leads to another important issue connected to anti-virus scanning. There is a constant flow of new viruses entering the Internet, hence the signature databases of the anti-virus scanners need to be continuously updated. There are heuristic scanning algorithms built into most of the modern scanners, meant to detect yet unknown breeds of viruses, but they tend to give a rather high number of both false positives and false negatives.

Also the softwares constituting the computer system have to be updated with the latest patches. This has to be done as soon as possible after the release of a new patch. There existed patches of the vulnerabilities used by the worms CodeRed and Nimda months before the two worms showed up. If the patches had been installed on the affected systems, the worms would

not have been able to spread as fast and far as they did. [75]

A complement to an anti-virus scanner is an integrity checker, which calculates checksums (or hashes) of files in the system. To be usable in practice only those files not subject to frequent updates or changes will be included in the check, because each time a checksum is calculated the file has to be verified to be clean (the update of the checksum to be authorised) by a user. Consequently the user has to be aware of the eventual implications of the action.

By using an integrity checker the changing or infection of a file is possible to detect, but it really is a reactive function. If the period between the change of the file and the detection is long, for example a virus may have been able to do considerable damage.

Another problem with using checksums is the fact that they are surjective, i.e. several files can have the same checksum. Thus a virus can (at least in theory) infect a file and pad it so the checksum is not changed when checked. Another way for a virus to defeat an integrity checker is to simply catch the call from the checksum calculation function and supply the function with an unchanged copy of the file.

By having a single point of connection to the outside world, or at least as few as possible, the risk of intrusion is decreased. Of course all the connections need to be properly secured by firewalls and e-mail filters with anti-virus scanning capabilities.

One weak point in the perimeter defence is the connection of remote users. Often they use laptops which are connected through modem connections, which to be fairly secure have to use encryption and good authentication routines. Also the users must keep a strict watch over their computers to prevent anyone from stealing them or maybe install a trojan horse program.

The above mentioned security enhancing methods will not give a 100% effective protection of a computer system. Therefore it is vital to back up the data in the system as often as possible. If anything happens and a back up has to be restored, every change to the data files in system since the time of the back up will be lost. Multiply the amount of work erased by the back up with the number of affected users (and customers) and the price will be high for not backing up more regularly. If then the cost of the bad-will and eventually severed customer relations are added to the result, the company might be on the brink of bankruptcy.

6.2 How a taxonomy increases security

The use of a taxonomy of software weapons in the field of computer security will improve the security related work in many ways. First of all, a taxonomy structures the field it is used in, both the general understanding of the field, as well as the more detailed knowledge of each entity contained. By knowing both on a macro level and a micro level what types of weapons there are and their technical characteristics, the suitable counter-

measures against each type can be stated in advance and easily found when needed.

Of course, the first thing to do when a system is hit is to find out what type of software weapon is used for the attack. First when that has been done different countermeasures can be applied. The phase of finding and identifying the used weapon might be made more effecive by using the categories of the proposed taxonomy as guidelines to what to look for. When the weapon has been identified, the taxonomy will point to other weapons with similar characteristics and therefore also indirectly to possible countermeasures.

The use of the taxonomy as a common ground for categorising software weapons will also make sharing new methods of defence easier. The joining of forces against an attacker is facilitated, because everyone involved in the defence instantly knows the characteristics of the weapon or weapons used.

Also the definition of the terms used need to be based on a common ground, because it will increase the mutual understanding and coordiantion of the defensive work. And the sooner the new definitions are decided upon, the better, because the present situation with a mismatch of different definitions leads to a significant risk of disastrous misunderstandings. The following fictive example will function as a scenario of how things can go wrong when not having a common ground for the used vocabulary. For example the term *trojan horse* is by some defined as a subclass of the *virus* and *worm* classes (see Section 4.6), while others define the relationship between the three terms the other way around (see Section 4.7). Now to the scenario:

> A small company has been hit by a virus packed into a downloaded shareware functioning as a trojan horse. The manager, regarding a trojan horse as a subclass of a virus, tells his employee, which regards a virus being a subclass of a trojan horse, that they have been hit by a trojan horse and that he needs to take appropriate measures to get rid of it. The employee deletes the file the manager has pointed out and also, as an extra precaution, reboots the system to get rid of active processes related to the trojan horse. He regards his actions to be sufficient, because if it had been a virus his manager would had said so. After all, he thinks, viruses are only a subclass of trojan horses. Thus the trojan horse cannot be parasitic.
>
> The manager on his side thinks he has been clear enough, trojan horses are a subclass of viruses and worms and therefore the employee should have understood that the software weapon might have been parasitic. Hence they both rest assure of that everything is taken care of. To really be on the sure side the employee makes a new backup of the whole computer system on the same tape as the old one, thinking the old back

> up is not needed anymore, because he has done what the manager told him to do. They are both up for a rather unpleasant surprise the next morning.

If they instead had used terms defined from a common ground, such as this proposed taxonomy of software weapons, they both would have known what the other meant. The employee would have restored a (clean) backup or gotten rid of the viral code from the infected files in another way.

A real life example of such a weapon as the one described in the scenario might be for example Sircam. It is often referred to as a worm, but it sends itself attached to an e-mail inserted into an ordinary file (which needs to be actively opened by the recipient) and also infects other computers via self-standing files transferred over network shares. It therefore would fit also the definitions of both a trojan horse and a worm.

Using a classification done with the help of the proposed taxonomy of software weapons, the employee would have known that the weapon was both parasitic and non-parasitic. He would probably also had had a whole class of similar weapons and the suitable countermeasures conneced to them to refer to.

By having a complete map of all existing software weapons and their realtionship through shared technical characteristics, the different defensive methods developed may be compared and possible weak spots found. Also the development of new countermeasuress might be made more effective and new fields of research found. A good example from another field is the periodic table, which gives much information on the different elements it contains, their number of electrons, their weight and chemical properties, if they are inert, etc. It has also helped researchers to find new elements, which in the end have resulted in several Nobel Prizes.

6.3 In the future

As stated in the title of this thesis this is a proposed taxonomy and it therefore has been subject to rather big and frequent changes, and still is, although the frequency is decreasing.

As mentioned in [3, 4] the taxonomy needs to be evaluated further. First of all the different categories and their alternatives have to be tested on a wider range of software weapons than what mostly is regarded as malware. The evaluation must be made independently by several people categorising the same weapons, using the same technical descriptions of the weapons. The result and their opinions then have to be compared and possibly changes have to be made to the taxonomy.

If the resulting categorisations are alike and no grave errors have been found, the formation of a reference database can begin. There already exists such a project for viruses in the form of the WildList [27], but that project is for natural reasons not done in accordance with this proposed

taxonomy. Therefore a new project has to be started, including *all* software weapons.

At the same time the computer security research community needs to jointly agree on definitions of the terms in the nomenclature used. Old terms might need to be redefined and new terms invented to extend the vocabulary to include different forms of non-viral software weapons. To simplify the work, this proposed taxonomy may be used as a common basis. Using the taxonomy will also guarantee that all terms are comparable regarding their technical characteristics and thus they may be arranged in a hierarchy reflecting their relationships.

Another issue needing to be dealt with is how to measure the characteristics of a weapon when no source code is available. This has only been mentioned briefly in the text, because it falls outside the scope of the thesis, but it still is an obstacle on the road towards a fully workable taxonomy and thus needs to be solved. The preferable solution is to find ways to recreate the source code in all possible situations.

6.4 Summary

There is definitely a need for a taxonomy of software weapons. As it is today the research might be hampered by the lack of generally accepted definitions of the terms used. Several taxonomies containing parts dealing with software weapons exist and a selection of them have been evaluated in this thesis. Unfortunately none of them fulfils all the requirements of a proper taxonomy, or the needs of the computer security field of research specified in Section 3.1.4 and Section 3.2.2.1.

The theories governing the formation of a taxonomy have been around for a while. A proper taxonomy should have mutually exclusive, exhaustive, and unambiguous categories. It also has to be formulated in a language suited for the intended readers, in other words both technically correct and yet rather general.

If a taxonomy meets these requirements it might be usable, but it also has to have a purpose. Regarding a taxonomy of software weapons there is a need for a detailed categorisation scheme able to classify the complete set of software weapons, both now and in the foreseeable future. The taxonomy also has to be used together with a definition of the entities in the field in question. The definition then works as a filter excluding all softwares not intended to be classified by the taxonomy.

The proposed taxonomy presented in this thesis is formulated without using any of the existing malware classification schemes as a base. Instead it is built from scratch, based exclusively on the technical characteristics of software weapons. In this way the taxonomy has not inherited any of the weaknesses of the current classification schemes. Hence, it has the potential to be used as a standard both for categorising weapons as well as redefining the terminology in the field.

To complete the taxonomy a new definition of software weapons is

formulated. The definition is not based on the intention of a user, creator or any other subjective and immeasurable property. Instead it is based solely on the weapon itself, by looking at the code. Although there might be practical problems with finding the code and reading it, these are solvable problems.

The taxonomy consists of 15 different and independent categories each having at least two alternatives, which together form a partitioning of the category. There have been some problems related to the definitions of some of the alternatives, but at the moment all of them are felt to be good and adequate.

Regarding the different countermeasures used against software weapons, there are (as in most cases) no perfect solutions. The best way is simply to keep the overall security as high as possible and to accept the fact that there always will be successful attacks. In that way the chance of noticing irregular behaviours in the computer system will be high, the users and the administrators will keep a good watch and not sit back and relax in false conviction that their system is impenetrable.

Finally, the taxonomy still needs to be tested further. Also the work with redefining the nomenclature of the field of research would need to be started.

7 Acronyms

API *Application Programming Interface*. The language or messaging format used by applications to communicate with an operating system, or other functions and programs. [76]

CAN *Candidate Number*. A vulnerability or exposure under investigation by the CVE Editorial Board for possible upgrading to a real vulnerability or exposure, when it also will receive a CVE number. [16]

CARO *Computer Antivirus Research Organization*. Created an unofficial recommendation of the naming procedure for viruses in 1991. An extension and update was proposed by Scheidl in 1999. [48]

CC *Common Criteria*. A joint effort by several governmental organisations, as a group called 'the Common Criteria Editing Board' [10, p. 159] to create a standard for evaluating and grading computer security. The effort is sponsored by the Common Criteria Project Sponsoring Organisations, which are listed in [51, p. ii].

CERT *Computer Emergency Response Team*. There are several national CERTs distributed over the world. They issue alerts and warnings regarding different computer security threats, as well as other security related publications. Their Coordination Center is situated at the Software Engineering Institute, Carnegie Mellon University. [77]

CVE *Common Vulnerabilities and Exposures*. A meta-base containing information of different vulnerabilities and exposures. The meta-base is supported by MITRE (a name, not an acronym). [14, 13]

DDoS *Distributed Denial of Service*. An attack degrading the availability of a computer system. The attack is executed using several remotely controlled agents all concurrently performing a denial of service attack on a specific target. [78, 79, 80]

DoS *Denial of Service*. An attack degrading the availability of a system by flooding it with a vast amount of network traffic or bad packets. [81]

EICAR *European Institute for Computer Antivirus Research*. However, the acronym has become self-standing and they also have expanded their working field to general IT security, with a focus on anti-virus research. [82]

FAT *File Allocation Table*. A table containing the addresses of the sectors on a harddisk occupied by the different files in a file system. The table is maintained by the operating system. [83]

FOI *Swedish Defence Research Agency*. In Swedish: 'Totalförsvarets forskningsinstitut'. [84]

ICAT *ICAT*. No extension has been found. Maybe it is not an acronym, but a name. [17]

IDS *Intrusion Detection System*. A system to monitor the activity in a network and in that way possibly detect intrusions. There are two methods used; rule-based monitoring and statistical-based. [76]

IIS *Internet Information Services*. A web-server from Microsoft. [85] It is widely used and therefore also often attacked.

ITSEC *Information Technology Security Evaluation Criteria*. A European document providing a framework for security evaluation of computer systems. [10, p. 155]

JVM *Java Virtual Machine*. It is like a virtual CPU that compiled Java programs run on. In this way the same Java code can be executed on several different computer platforms, without having to be recompiled. [76]

NAT *Network Address Translation*. A NAT server offers the possibility to hide the internal addresses of a network and thus only have one IP address visible to the outside world. This service is also offered by for example a proxy server. [76, 83]

NCW *Network Centric Warfare*. The new idea of how to revolutionise the way of faring war. All different participants are meant to be connected in a network. In this way the troops may be better coordinated, information sharing among the soldiers and their superiors will be facilitated, and more can be achieved by less participants. [86]

NIST *(US) National Institute of Standards and Technology*. The goal of this federal agency is to strengthen and advance the science and technology within USA. [76, 83]

NordSec 2002 *The 7th Nordic Workshop on Secure IT Systems*. The workshop was held at Karlstad University, Sweden 7–8 November 2002. [87]

RMI *Remote Method Invocation*. A protocol developed by Sun to allow Java objects to remotely communicate with other Java objects over a network. [76, 83]

RPC *Remote Procedure Call*. A protocol (middleware) that allows a computer to execute a program on another (server) computer. [76, 83]

TFN2K *Tribe Flood Network 2000*. A DDoS software weapon. [60]

VBA *Visual Basic (for) Applications*. A programming language based on `BASIC` and developed by Microsoft. It provides a graphical programming environment. [83]

VTC *Virus Test Center*. A laboratory specialised in reverse engineering and decompilation of viruses. They also perform tests of the efficiency of different virus-scanners on the market. The laboratory is headed by professor Klaus Brunnstein and is part of the Computer Science Department at the University of Hamburg, Germany. [88]

References

[1] Michael Best, *Printing, and problems in Shakespeare's text*, September 2001.
`http://web.uvic.ca/shakespeare/Library/SLTnoframes/literature/problems.html`, accessed 9 December 2002.

[2] William Shakespeare, *The Arden Shakespeare Complete Works*, chapter Romeo and Juliet, pp. 1005–1038, Thomas Nelson and Sons Ltd, Walton-on-Thames, Surrey, UK, 1998.

[3] Martin Karresand, 'TEBIT – Teknisk Beskrivningsmodell för IT-vapen (TEBIT. Technical characteristics' description model for IT-weapons)', Tech. Rep. FOI-R–0305–SE (Metodrapport/Methodology report), Command and Control Warfare Technology, FOI - Swedish Defence Research Agency, Linköping, Sweden, August 2001.

[4] Martin Karresand, 'A Proposed Taxonomy for IT Weapons', in *Nord-Sec 2002 – Proceedings of the 7th Nordic Workshop on Secure IT Systems*, Simone Fisher-Hübner and Erland Jonsson, Eds., Karlstad, Sweden, November 2002, pp. 244–260, Karlstad University Studies.

[5] Ian Whalley, Bill Arnold, David Chess, John Morar, Alla Segal, and Morton Swimmer, *An Environment for Controlled Worm Replication and Analysis or: Internet-inna-Box*, September 2000.
`http://www.research.ibm.com/antivirus/SciPapers/VB2000INW.htm`, accessed 18 July 2002.

[6] Ian Whalley, *Testing Times for Trojans*, October 1999.
`http://www.research.ibm.com/antivirus/SciPapers/Whalley/inwVB99.html`, accessed 18 July 2002.

[7] CERT (Computer Emergency Response Team), *CERT Advisory CA-1995-06 Security Administrator Tool for Analyzing Networks (SATAN)*, April 1995.
`http://www.cert.org/advisories/CA-1995-06.html`, accessed 12 June 2002.

[8] Sarah Gordon, *Devil's Advocate*, 1995.
`http://www.commandsoftware.com/virus/satan.html`, accessed 23 July 2002.

[9] CIAC (Computer Incidents Advisory Center), *Information Bulletin F-20: Security Administrator Tool for Analyzing Networks (SATAN)*, April 1995.

http://www.ciac.org/ciac/bulletins/f-20.shtml, accessed 12 June 2002.

[10] Dieter Gollmann, *Computer Security*, John Wiley & Sons, 1999, ISBN 0-471-97844-2.

[11] Carl E Landwehr, Alan R Bull, John P McDermott, and William S Choi, 'A Taxonomy of Computer Program Security Flaws, with Examples', *ACM Computing Surveys*, vol. 26, no. 3, September 1994.
http://chacs.nrl.navy.mil/publications/CHACS/1994/1994landwehr-acmcs.pdf, accessed 12 June 2002.
A note taken from the text published on the web: 'As revised for publication in ACM Computing Surveys 26, 3 (Sept., 1994). This version, prepared for electronic distribution, reflects final revisions by the authors but does not incorporate Computing Surveys´ copy editing. It therefore resembles, but differs in minor details, from the published version. The figures, which have been redrawn for electronic distribution are slightly less precise, pagination differs, and Table 1 has been adjusted to reflect this'.

[12] CVE,
http://cve.mitre.org/about/terminology.html, accessed 4 July 2002.

[13] CVE,
http://cve.mitre.org/about/index.html, accessed 24 June 2002.

[14] MITRE, *The Early Years*.
http://www.mitre.org/about/history.shtml, accessed 12 June 2002.

[15] Daniel L Lough, *A Taxonomy of Computer Attacks with Applications to Wireless Networks*, PhD thesis, Virgina Polytechnic Institute and State University, April 2001.
http://scholar.lib.vt.edu/theses/available/etd-04252001-234145/unrestricted/lough.dissertation.pdf, accessed 13 June 2002.

[16] CVE,
http://cve.mitre.org/docs/docs2000/naming_process.html, accessed 12 June 2002.

[17] ICAT,
http://icat.nist.gov/icat.cfm, accessed 12 June 2002.

[18] ICAT,
http://icat.nist.gov/icat_documentation.htm, accessed 27 September 2002.

[19] Simon Hornblower and Tony Spawforth, Eds., *Who's Who in the Classical World*, Oxford University Press, 2000, ISBN 0-19-280107-4.

[20] Encyclopedia Britannica Online, `http://www.britannica.com/eb/article?eu=119735&tocid=48695`, accessed 12 June 2002.

[21] John D Howard and Thomas A Longstaff, *A Common Language for Computer Security Incidents*, Sandia National Laboratories, Livermore, CA, October 1998. `http://www.cert.org/research/taxonomy_988667.pdf`, accessed 6 September 2002.

[22] Ivan V Krsul, *Software Vulnerability Analysis*, PhD thesis, Purdue University, May 1998. `http://www.acis.ufl.edu/~ivan/articles/main.pdf`, accessed 13 June 2002.

[23] George Gaylord Simpson, 'The principles of classification and a classification of mammals', in *Bulletin of the American Museum of Natural History*, vol. 85, pp. 1–350. 1945.

[24] Ulf Lindqvist, *On the Fundamentals of Analysis and Detection of Computer Misuse*, PhD thesis, Department of Computer Engineering, Chalmers University of Technology, Göteborg, Sweden, 1999. `http://www.ce.chalmers.se/staff/ulfl/pubs/ul-phd.pdf`, accessed 11 October 2002.

[25] Jakub Kaminski and Hamish O'Dea, *How to smell a RAT - remote administration tools vs backdoor Trojans*. `http://www.virusbtn.com/conference/this_year/abstracts/remote_administration.xml`, accessed 22 July 2002.
Only the abstract of the paper was available and therefore no references are made to the body of the document.

[26] David Moore, Geoffrey M Voelker, and Stefan Savage, *Inferring Internet Denial-of-Service Activity*, 2001. `http://www.caida.org/outreach/papers/2001/BackScatter/usenixsecurity01.pdf`, accessed 22 November 2002.

[27] Joe Wells, *How Scientific Naming Works*. `http://www.wildlist.org/naming.htm`, accessed 22 November 2002.

[28] Per Ånäs, 'Aktören vid IT-relaterade attacker – vem, varför och hur? (The actor making IT-related attacks - who, why and how?)', Tech.

Rep. FOI-R–0271–SE (Underlagsrapport/Base data report), Defence Analysis, Swedish Defence Research Agency, December 2001.

[29] Bernie Klinder, *Computer Virus and Malware Primer for Network Administrators*, September 2002.
`http://www.labmice.net/AntiVirus/articles/avprimer.htm`, accessed 22 November 2002.

[30] Vesselin Vladimirov Bontchev, *Methodology of Computer Anti-Virus Research*, PhD thesis, University of Hamburg, Germany, 1998.

[31] Marko Helenius, *A System to Support the Analysis of Antivirus Products' Virus Detection Capabilities*, PhD thesis, University of Tampere, Finland, 2002.
`http://acta.uta.fi/pdf/951-44-5394-8.pdf`, accessed 22 July 2002.

[32] *Oxford English Dictionary Online*, 2nd edition, 1989.
`http://dictionary.oed.com/cgi/entry/00247833`, accessed 24 November 2002.

[33] Sybil P Parker, Ed., *Dictionary of bioscience*, McGraw-Hill, 1997, ISBN 0-07-114919-8.

[34] Ulf Lindqvist and Erland Jonsson, 'How to Systematically Classify Computer Security Intrusions', in *Proceedings of the 1997 IEEE Symposium on Security & Privacy*, Oakland, CA, 1997, pp. 154–163, IEEE Computer Society Press.
`http://www.ce.chalmers.se/staff/ulfl/pubs/sp97ul.pdf`, accessed 12 June 2002.

[35] John D Howard, *An Analysis of Security Incidents on the Internet 1989-1995*, PhD thesis, Carnegie Mellon University, Pittsburg, April 1997.
`http://www.cert.org/research/JHThesis/Word6/`, accessed 12 June 2002.

[36] Richard Ford, *Malware*.
`http://www.malware.org/malware.htm`, accessed 17 July 2002.

[37] Morton Swimmer, *Malware*.
`http://www.swimmer.org/morton/malware.html`, accessed 18 July 2002.

[38] Fred Cohen, 'Computer viruses: Theory and experiments', *Computers & Security*, vol. 6, no. 1, pp. 22–35, February 1987.

[39] David M Chess and Steve R White, *An Undetectable Computer Virus*, IBM Thomas J. Watson Research Center, Hawthorne, New York, USA, 2000.
`http://www.research.ibm.com/antivirus/SciPapers/VB2000DC.pdf`, accessed 21 November 2002.

[40] David G Boney, 'The Plague: An Army of Software Agents for Information Warfare', Tech. Rep., Department of Computer Science, School of Engineering and Applied Science, Washington D.C. 20052, June 1999.

[41] Lance J Hoffman, Ed., *Rogue Programs: Viruses, Worms, and Trojan Horses*, Van Nostrand Reinhold, 1990.

[42] Christopher V Feudo, *The Computer Virus Desk Reference*, Business One Irwin Computer, 1992.

[43] Klaus Brunnstein, *From AntiVirus to AntiMalware Software and Beyond: Another Approach to the Protection of Customers from Dysfunctional System Behaviour*, Faculty for Informatics, University of Hamburg, Germany, July 1999.
`http://csrc.nist.gov/nissc/1999/proceeding/papers/p12.pdf`, accessed 22 July 2002.

[44] Jo Ticehurst, *Virus naming chaos causes confusion*, October 2000.
`http://newsletter.vnunet.com/News/1112012`, accessed 18 November 2002.

[45] Ken Dunham, *Malware Taxonomy: Challenges*, July 2001.
This text is the last part of three in a series published at SecurityPortal in July, 2001. The other two are called *The Great Analogy* and *Malware Taxonomy: Introduction*. Unfortunately the portal is nowadays offline. The article was received by e-mail from Ken Dunham, `dunhamk@rmci.net`.

[46] Ian Whalley, *VGrep*, Advanced Security Research, McAfee Security.
`http://toronto.virusbtn.com/resources/vgrep/`, accessed 21 November 2002.

[47] Fridrik Skulason and Vesselin Bontchev, *A New Virus Naming Convention*, 1991.
`http://vx.netlux.org/lib/asb01.html`, accessed 18 November 2002.

[48] Gerald Scheidl, *Virus Naming Convention 1999 (VNC99)*, beta edition, July 1999.
`http://members.chello.at/erikajo/vnc99b2.txt`, accessed 22 October 2002.

[49] Donn B Parker, *Advancing Security*, 1999.
http://www.infosecuritymag.com/articles/1999/parker2.shtml, accessed 22 November 2002.

[50] Winn Schwartau, *The Basics are the Basics*, September 1999.
http://www.infowar.com/chezwinn/articles092899/TheBasicsAreTheBasics.shtml, accessed 22 November 2002.

[51] Common Criteria, *Common Criteria for Information Technology Security Evaluation, part 1*, 2.1 edition, August 1999.
http://www.commoncriteria.org/docs/PDF/CCPART1V21.PDF, accessed 3 December 2002.

[52] Philip Babcock Gove, Ed., *Webster's Third New International Dictionary*, Merriam-Webster Inc., 1993, ISBN 3-8290-5292-8.

[53] Eric Chien, *CodeRed Worm*, Symantec, July 2002.
http://securityresponse.symantec.com/avcenter/venc/data/codered.worm.html, accessed 24 July 2002.

[54] Trend Micro, *CODERED.A*, July 2001.
http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=CODERED.A&VSect=T, accessed 24 July 2002.

[55] eEye Digital Security, *.ida "Code Red" Worm*, July 2001.
http://www.eeye.com/html/Research/Advisories/AL20010717.html, accessed 13 September 2002.

[56] Digital Communication Technologies Inc, *Lightfoot 32-bit Java Processor Core*, August 2001.
http://www.dctl.com/downloads/fpga_lightfoot_ds.pdf, accessed 6 December 2002.

[57] aJile Systems Inc, *Real-time Low-power $Java_TM$ Processor aJ-80*, 2000.
http://www.ajile.com/downloads/aj80.pdf, accessed 6 December 2002.

[58] Sun Microsystems, *Sun Unveils Its First Java Processor microJava701 Looks to Post Industry's Highest Caffeinemarks*, October 1997.
http://www.sun.com/smi/Press/sunflash/9710/sunflash.971015.1.html, accessed 6 December 2002.

[59] Peter Ferrie, *W32.Peelf.2132*, April 2002.
http://securityresponse.symantec.com/

avcenter/venc/data/w32.peelf.2132.html, accessed 6 December 2002.

[60] Jason Barlow and Woody Thrower, *TFN2K – An Analysis*, March 2000. http://packetstormsecurity.nl/distributed/TFN2k_Analysis-1.3.txt, accessed 17 November 2002.

[61] David Dittrich, George Weaver, Sven Dietrich, and Neil Long, *The "mstream" distributed denial of service attack tool*, May 2000. http://staff.washington.edu/dittrich/misc/mstream.analysis.txt, accessed 16 November 2002.

[62] David Dittrich, *The "stacheldraht" distributed denial of service attack tool*, December 1999. http://staff.washington.edu/dittrich/misc/stacheldraht.analysis, accessed 24 July 2002.

[63] David Dittrich, *The DoS Project's "trinoo" distributed denial of service attack tool*, October 1999. http://staff.washington.edu/dittrich/misc/trinoo.analysis.txt, accessed 24 July 2002.

[64] David Dittrich, *The "Tribe Flood Network" distributed denial of service attack tool*, October 1999. http://staff.washington.edu/dittrich/misc/tfn.analysis.txt, accessed 16 November 2002.

[65] Eric Chien and Peter Szor, *CodeRed II*, Symantec, July 2002. http://securityresponse.symantec.com/avcenter/venc/data/codered.ii.html, accessed 18 October 2002.

[66] Trend Micro, *CODERED.C*, August 2001. http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=CODERED.C&VSect=T, accessed 18 October 2002.

[67] eEye Digital Security, *CodeRedII Worm Analysis*, August 2001. http://www.eeye.com/html/Research/Advisories/AL20010804.html, accessed 18 October 2002.

[68] Eric Chien, *W32.Nimda.A@mm*, Symantec, July 2002. http://securityresponse.symantec.com/avcenter/venc/data/w32.nimda.a@mm.html#technicaldetails, accessed 21 October 2002.

[69] K Tocheva, G Erdelyi, A Podrezov, S Rautiainen, and M Hypponen, *Nimda*, F-Secure, September 2001.

http://www.europe.f-secure.com/v-descs/nimda.shtml, accessed 21 October 2002.

[70] Trend Micro, *PE_NIMDA.A*, October 2001. http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?Vname=PE_NIMDA.A&VSect=T, accessed 21 October 2002.

[71] Peter Ferrie and Peter Szor, *W32.Sircam.Worm@mm*, Symantec, July 2002. http://securityresponse.symantec.com/avcenter/venc/data/w32.sircam.worm@mm.html#technicaldetails, accessed 23 October 2002.

[72] Gergely Erdelyi and Alexey Podrezov, *Sircam*, F-Secure, July 2001. http://www.europe.f-secure.com/v-descs/sircam.shtml, accessed 23 October 2002.

[73] Trend Micro, *WORM_SIRCAM.A*, October 2001. http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=WORM_SIRCAM.A&VSect=T, accessed 23 October 2002.

[74] Terrance A Roebuck, *A Holistic Approach to Viruses and Other Malware*. http://abyss.usask.ca/~roebuck/malware.HTML, accessed 26 November 2002.

[75] Todd McGuiness, *Defense In Depth*, November 2001. http://rr.sans.org/securitybasics/defense.php, accessed 27 November 2002.

[76] Tom Sheldon, *McGraw-Hill Encyclopedia of Networking & Telecommunications*, McGraw-Hill, Berkeley, California 94710, USA, 2001, ISBN 0-07-212005-3.

[77] CERT/CC, *Welcome*, December 2002. http://www.cert.org, accessed 9 December 2002.

[78] Tim Yardley, *Distributed Attacks and the Way To Deal With Them*, 1999. http://packetstormsecurity.com/papers/contest/Tim_Yardley.doc, accessed 9 December 2002.

[79] CERT/CC, *Results of the Distributed-Systems Intruder Tools Workshop*, December 1999. http://www.cert.org/reports/dsit_workshop-final.html, accessed 9 December 2002.

[80] Serghei Sevcenco, *Distributed Denial of Service (DDoS) attacks*, Symantec, July 2001.
http://securityresponse.symantec.com/avcenter/venc/data/ddos.attacks.html, accessed 10 December 2002.

[81] CERT® Coordination Center, *Denial of Service Attacks*, June 2001.
http://www.cert.org/tech_tips/denial_of_service.html, accessed 9 December 2002.

[82] *EICAR Online*, 2002.
http://www.eicar.org, accessed 9 December 2002.

[83] Thomas M Thomas, II, *Thomas' Concise Telecom & Networking Dictionary*, McGraw-Hill, Berkeley, California 94710, USA, 2000, ISBN 0-07-212253-6.

[84] *Welcome to FOI – The Swedish Defence Research Agency*, September 2002.
http://www.foi.se/english/index.html, accessed 9 December 2002.

[85] Microsoft, *Internet Information Services Community Center*.
http://www.microsoft.com/windows2000/community/centers/iis/default.asp, accessed 10 December 2002.

[86] David S Alberts and John J Garstka, *Network Centric Warfare – Department of Defense Report to Congress*.
http://www.c3i.osd.mil/NCW/, accessed 9 December 2002.
The text that was read for this reference was the 'Executive Summary' found at http://www.c3i.osd.mil/NCW/ncw_exec_sum.pdf. However, the general URL stated above leads to the main website, which contains the (complete?) report in several large PDF files.

[87] Albin Zuccato, *Nordsec 2002*, July 2002.
http://www.cs.kau.se/nordsec2002/index.html, accessed 9 December 2002.

[88] Klaus Brunnstein, *Virus Test Center*, April 2002.
http://agn-www.informatik.uni-hamburg.de/vtc/, accessed 9 December 2002.

Appendix A

The NordSec 2002 paper

A Proposed Taxonomy for IT Weapons*

Martin Karresand

FOI

Swedish Defence Research Agency, Division of Command and Control Systems,

Department of Systems Analysis and IT Security

Box 1165, SE-581 11 Linköping, Sweden

martin.karresand@foi.se

Abstract

This report presents a proposal for a taxonomy of IT weapons, limited to computer software. Because of this limitation the term software weapons is used instead of IT weapons. A definition of software weapons is also formulated. No other taxonomy with the above scope is known to exist today. This taxonomy therefore offers a theoretical base for the unification of the nomenclature for classification of software weapons.

The taxonomy contains 15 categories of general properties of a software weapon. It has been adapted to international standards through a connection to the CVE list (Common Vulnerabilities and Exposures), which is maintained by MITRE.

The problem of how to make unambiguous classifications of combined software weapons is discussed and a solution is proposed. Each category of the taxonomy is explained in a separate paragraph. Thereafter the taxonomy is used to classify two well known software weapons.

Keywords: computer security, information warfare, IT weapon, IW, malware, software weapon, taxonomy, virus, worm.

1 Introduction

The terminology used in the IT security area is not yet fully standardised and the situation is getting worse [1, 2, 3] because of the continuous influx of new

*To appear in: Nordsec 2002 – Proceedings of the 7th Nordic Workshop on Secure IT Systems, Karlstad University, Sweden, 7–8 November 2002.

members to the research community, who all have their own preferred vocabulary. Hence there is an increasing need for a standardisation of the used terms.

On top of that the research being done so far has been concentrated on the pragmatic, technical side of the spectrum, i.e. ways of detecting the increasing amount of malware (malicious software) being written. The classification and grouping of the malware has been given less attention and the area is therefore hard to take in.

To enable the development of efficient countermeasures there is a need for an unambiguous classification of the software used to cause harm to individuals and organisations via computer systems. Also the users of the computer systems need to have a general understanding of the threats posed by different types of malware. This would lead to a higher degree of awareness of possible malware attacks and in that way higher security. One step towards this goal is to have commonly acknowledged names for the separate types of malware. Consequently these types must be well defined too.

Furthermore, the education of IT security personnel would benefit from a structured classification of the malware area. A common vocabulary would for example decrease the risk of misunderstandings.

Whether a specific tool would be classified as a weapon or not is often judged from the context of the situation where the tool is used. This is the juridical point of view, the tool a murderer used is to be regarded as a weapon, because he or she used it with the intent to kill. Consequently, anything can be a weapon.

The sentence 'He used a pillow as a weapon' gives that the pillow was a weapon *in that specific situation*. But by disregarding the context and just concentrate on the '*as a weapon*' part of the sentence, we see that a tool must have certain properties to be a weapon.[1] These properties are in some way measurable; they do harm (why else would they be used for fighting and attacking?). If the line of argument is transferred to the world of computers, the result is that a certain class of software has a specific set of properties, which are measurable, and those properties define the software as weapons.

The advantage of this approach is the much lesser degree of ambiguity. A weapon is a weapon because of its properties and as long as the purpose is to study it technically, that is enough. With a deeper knowledge of the technical details of software weapons (malware) as a group, they can be classified, properly named, etc. This in turn leads to a more structured knowledge of the area and thus a possibility to develop better defences, maybe even in advance. Or as Sun Tzu once wrote in his book The Art of War [5, chapter VI]:

> Whoever is first in the field and awaits the coming of the enemy,

[1] One definition is 'an object such as a knife, gun, bomb, etc. that is used for fighting or attacking sb'. [4]

> will be fresh for the fight; whoever is second in the field and has to hasten to battle will arrive exhausted.

1.1 Background

This is an updated version of a report [6] written in Swedish. The amendments were mostly related to preparing the report for international publishing. Some parts have also been rewritten because new background material has been found.

In an attempt to somewhat lessen the emotional charge in the used vocabulary, the term *software weapon* will be used throughout the text. Something malicious can be nothing but evil, but a weapon is simply a tool that has the ability to cause harm and that can be used in both offensive and defensive situations.

Regarding the area of malware, several definitions and classification schemes exist for different parts of the area (see for example [7, 8, 9]). Most of them deal with viruses and worms, and only casually mention other types of malicious software. They all give their own way of measuring, or quantifying, maliciousness and at the same time conclude that this cannot be done objectively.

No definition or taxonomy[2] covering the complete set of software-based IT weapons has yet been found by the author.

1.2 Purpose

The purpose of the report is to present a taxonomy of software weapons, and also give a definition supporting the taxonomy. The taxonomy and definition are not meant to be complete in any way, but merely suggestions for future work.

The purpose of the taxonomy is to fill the needs stated in the introduction, or at least lay the foundation for a future fulfilment of them.

1.3 Scope

The taxonomy only handles software based (IT) weapons from a technical point of view. Chipping[3] is considered to be hardware based and is therefore not discussed.

1.4 Method

The study has been done with a broad technical base. Several different types of material have been studied. Most of the material has been taken from the Internet to give up to date information. It has mostly been descriptions of tools and

[2]The word originates from the two Greek words *taxis*, arrangement, order, and *nomos*, distribution.

[3]Malicious alterations of computer hardware.

methods used by hackers. Also technical reports, dissertations, and taxonomies focused on IT security and malware have been used.

2 A Taxonomy of Software Weapons

My own hypothesis of why no other taxonomy of software weapons has yet been found can be summarised in the following points:

- The set of all software weapons is (at least in theory) infinite, because new combinations and strains are constantly evolving. Compared to the biological world, new mutations can be generated at light speed.
- It is hard to draw a line between administrative tools and software weapons. Thus it is hard to strictly define what a software weapon is.
- Often software weapons are a combination of other, atomic, software weapons. It is therefore difficult to unambiguously classify such a combined weapon.
- There is no unanimously accepted theoretical foundation to build a taxonomy on. For instance there are (at least) five different definitions of the term *worm* [10] and seven of *trojan* [11].
- By using the emotionally charged word *malicious* together with *intent*, the definitions have been crippled by the discussion whether to judge the programmer's or the user's intentions.

2.1 Theory

As a consequence of some of the problems mentioned above, the set of software weapons will grow continuously. Therefore it can be regarded as always new and unexplored. The fact that software weapons can be created from combinations of other software weapons, without limitations, gives that a traditional taxonomy based on relationships would not work very well. The rules for classification would grow indefinitely complex and soon get out of hand. A better solution would be to base the taxonomy on technical characteristics. With a proper selection of characteristics, such a taxonomy would have the potential to work for more than a few years.

It is not enough to find a working set of characteristics to get a good taxonomy, though. It must fulfil a few more requirements to be useful. Daniel Lough has created a list of 18 properties from 5 different taxonomies of IT security, which he presents in his dissertation [12]. I consider the following properties taken from

two of those taxonomies [13, 14] to be the most important. The categories of the taxonomy should:

- Be *mutually exclusive* and *exhaustive* so that the taxonomy completely covers the intended area, i.e. be a *partitioning* of the area
- Be *unambiguous* to prevent subjective interpretations
- *Usable* through the use of well known and consistent terminology.

To minimise the risk of subjective interpretations when classifying objects, the alternatives in each category should be based on measurable or observable characteristics [15]. In the case of software these characteristics are the instructions and algorithms constituting the software [16]. This will guarantee that the classification of a software weapon will be the same, regardless of who is classifying.

How the characteristics of the software weapon shall be found is a separate problem. It can be done by either having access to the source code of the weapon, or by re-engineering the source code from a binary version of the weapon. A third way is to have some sort of automatic analysis software; a virtual environment where the software weapon could be scientifically studied in a controlled manner. Such an environment already exists for worms and viruses [10].

2.2 Definition

In this section a definition of software weapons is presented, together with the reasons for developing it. To avoid influences from the definitions of malware mentioned earlier, the new definition has been constructed with information warfare as a base.

2.2.1 Background.

There are several definitions of IT and cyber warfare. Of course they cover a much larger area than just software weapons, but they do give a hint of what the important things are. The US Department of Defense has the following definition of the military part of information warfare [17]:

> Information Warfare - Actions taken to achieve information superiority in support of national military strategy by affecting adversary information and information systems while leveraging and defending our information and systems.

Dr. John Alger, MITRE Corporation, Enterprise Security Solutions Department, gives the following definition of information warfare in a book by Winn Schwartau [18, p. 12]:

> Information Warfare consists of those actions intended to protect, exploit, corrupt, deny, or destroy information or information resources in order to achieve a significant advantage, objective, or victory over an adversary.

A similar definition is given by Ivan Goldberg [19], head of IASIW (US Institute for the Advanced Study of Information Warfare):

> Information warfare is the offensive and defensive use of information and information systems to deny, exploit, corrupt, or destroy, an adversary's information, information-based processes, information systems and computer-based networks while protecting one's own. Such actions are designed to achieve advantages over military or business adversaries.

All the above definitions mentions that an advantage over an adversary should be achieved and this should be done by influencing the adversary's information systems. An advantage in the software context would correspond to a breach in the security of the adversary's computer system. The influencing part would then be the instructions of the tool(s) used for the attack. Thus a software weapon should have such properties.

The definitions mentioned above are all very much alike, which might indicate that they all have the same source. If so, three renowned institutions has adopted it, which in that case strengthens its importance. I therefore think that the definitions above carry such weight that they can be used as a basis for the definition of software weapons used in this report.

2.2.2 Preliminary Definition.

The preliminary definition of software weapons[4] used at FOI[5] has the following wording (translated from Swedish):

> [...] software for logically influencing information and/or processes in IT systems in order to cause damage.[6]

This definition satisfies the conditions mentioned earlier in the text. One thing worth mentioning is that tools without any logical influence on information or processes are not classified as software weapons by this definition. This means that for instance a sniffer is not a software weapon. Even a denial of service

[4]The term *IT weapon* is used in the report FOI report.

[5]Swedish Defence Research Agency

[6]In Swedish: '[...] programvara för att logiskt påverka information och/eller processer i IT-system för att åstadkomma skada.'

weapon might not be regarded as a weapon depending on the interpretation of 'logically influencing . . . processes'. A web browser on the other hand falls into the software weapon category, because it can be used in a dot-dot[7] attack on a web server and thus affect the attacked system logically.

Furthermore, the definition does not specify if it is the intention of the user or the programmer, that should constitute the (logical) influence causing damage. If it is the situation where the tool is used that decides whether the tool is a software weapon or not, theoretically all software ever produced can be classified as software weapons.

If instead it is the programmer's intentions that are decisive, the definition gives that the set of software weapons is a subset (if yet infinite) of the set of all possible software. But in this case we have to trust the programmer to give an honest answer (if we can figure out whom to ask) on what his or her intentions was.

A practical example of this dilemma is the software tool *SATAN*, which according to the creators was intended as a help for system administrators [20, 21]. *SATAN* is also regarded as a useful tool for penetrating computer systems [22]. Whether *SATAN* should be classified as a software weapon or not when using the FOI definition is therefore left to the reader to subjectively decide.

2.2.3 New Definition.

When a computer system is attacked, the attacker uses all options available to get the intended result. This implies that even tools made only for administration of the computer system can be used. In other words there is a grey area with powerful administrative tools, which are hard to decide whether they should be classified as software weapons or not. Hence a good definition of software weapons is hard to make, but it might be done by using a mathematical wording and building from a foundation of measurable characteristics.

With the help of the conclusions drawn from the definitions of information warfare the following suggestion for a definition of software weapons was formulated:

> A software weapon is software containing *instructions* that are necessary and sufficient for a *successful attack* on a *computer system*.

Even if the aim was to keep the definition as mathematical as possible, the natural language format might induce ambiguities. Therefore a few of the terms

[7] A dot-dot attack is performed by adding two dots directly after a URL in the address field of the web browser. If the attacked web server is not properly configured, this might give the attacker access to a higher level in the file structure on the server and in that way non-authorised rights in the system.

used will be further discussed in separate paragraphs.

Since it is a definition of *software* weapons, manual input of instructions is excluded.

Instructions. It is the instructions and algorithms the software is made of that should be evaluated, not the programmer's or the user's intentions. The instructions constituting a software weapon must also be of such dignity that they together actually will allow a breakage of the security of an attacked system.

Successful. There must be at least one computer system that is vulnerable to the tool used for an attack, for the tool to be classified as a software weapon. It is rather obvious that a weapon must have the ability to do harm (to break the computer security) to be called a weapon. Even if the vulnerability used by the tool might not yet exist in any working computer system, the weapon can still be regarded as a weapon, as long as there is a theoretically proved vulnerability that can be exploited.

Attack. An attack implies that a computer program in some way affects the *confidentiality*[8], *integrity*[9] or *availability*[10] of the attacked computer system. These three terms form the core of the continually discussed formulation of computer security. Until any of the suggested alternatives is generally accepted, the definition of attack will adhere to the core.

The security breach can for example be achieved through taking advantage of flaws in the attacked computer system, or by neutralising or circumventing its security functions in any way.

The term *flaw* used above is not unambiguously defined in the field of IT security. Carl E Landwehr gives the following definition [24, p. 2]:

> [...] a security flaw is a part of a program that can cause the system to violate its security requirements.

Another rather general, but yet functional, definition of ways of attacking computer systems is the definition of vulnerability and exposure [25] made by the CVE[11] Editorial Board.

[8] '[P]revention of unauthorised disclosure of information.'[23, p. 5]

[9] '[P]revention of unauthorised modification of information.'[23, p. 5]

[10] '[P]revention of unauthorised withholding of information or resources.'[23, p. 5]

[11] '[CVE is a] list of standardized names for vulnerabilities and other information security exposures – CVE aims to standardize the names for all publicly known vulnerabilities and security exposures. [...] The goal of CVE is to make it easier to share data across separate vulnerability databases and security weapons.' [26]. The list is maintained by MITRE [27].

Computer System. The term computer system embraces all kinds of (electronic)[12] machines that are programmable and all software and data they contain. It can be everything from integrated circuits to civil and military systems (including the networks connecting them).

2.2.4 Evaluation.

To test if the new definition has the intended extent, it is applied to a selection of common hacker tools. First five classes of tools chosen from a list made by David Icove [28, pp. 29–60] is used, then two tools not commonly regarded as software weapons, a web browser and a word processor.

To get as relevant a test as possible, tools that have a high ambiguity with respect to whether they should be regarded as software weapons or not are selected.

Denial of Service. Tools that in some way degrade the service of a computer system exist in several versions. The instructions of such a tool is both necessary and sufficient to successfully degrade the availability of the attacked system and it is thus a software weapon.

Data Diddling. A tool performing unauthorised manipulation of data on the attacked system can for instance be a log eraser. The definition states that this is a software weapon, because the tool affects the integrity of the attacked system.

Port Scanning. A port scan can be compared to going round a house (in full daylight) trying all the doors and windows to see if any of them is open [29]. Such knowledge can then be used for intrusion.

On the other hand, merely studying the visual characteristics of an object does not affect its confidentiality. Something clearly visible cannot be regarded as secret. Thus, such a simple port scanner as the one described above is not sufficient enough to affect the confidentiality of the scanned system and is therefore not a software weapon.

However, what today commonly is known as a security scanner is more powerful than the tool described above. A few examples are *SATAN*, *Nessus*, and *NSS*. They can for instance search for specific vulnerabilities and perform port mapping for different applications. Such a tool contains instructions both necessary and sufficient to affect the confidentiality of the attacked system.

[12] This term might be to restrictive. Already advanced research is done in for example the areas of biological and quantum computers.

Password Sniffing. By analysing the content of packets sent over a network passwords can be found, without interrupting the network traffic. If the sniffed passwords are unencrypted (or can be decrypted by the sniffer), the password sniffing is necessary and sufficient to violate the confidentiality of the attacked system and the sniffer is therefore a software weapon.

On the other hand, if the sniffer tool itself does merely send the encrypted passwords to another tool for decryption, its instructions is not sufficient for a successful attack. In other words, a sniffer is a good example of a tool that reside in the grey area.

Traffic Analysis. Tools performing traffic analysis work in a similar way to password sniffers, but instead they use the address field of the data packet. In that way they can be used for mapping the topology of a network. The information can be used to identify specific servers and security functions. These can then be circumvented or attacked.

The situation can be compared to a reconnaissance device collecting data on the positions of enemy troops on a battle field. Such data is most likely confidential and might be necessary and sufficient for a successful attack on the enemy, i.e. a traffic analysis tool is a software weapon [30, 29]

However, many traffic analysis tools are manually operated, i.e. the user gives the parameters that control the operation. These parameters can then be viewed as the instructions that perform the actual attack. Thus in this case the traffic analysis tool itself cannot be regarded as being a software weapon. Instead it should be compared to a terminal program.

From the above we can see that a traffic analyser occupies the grey area mentioned before. Each traffic analyser therefore has to be inspected separately to determine whether it should be classified as a software weapon or not.

Web Browser. Using a web browser a hacker can make a dot-dot attack on a computer system. In this case the actual instructions representing the attack are given by the user, not the web browser. Thus the instructions constituting a web browser are not sufficient to successfully attack a computer system and consequently the browser is not a software weapon. Instead it can be regarded as a manually operated terminal program.

Word Processor. Through the built-in macro language, a word processor can be utilised to perform unauthorised actions on an attacked system. The instructions used for the attack are given by the macro, the word processor only interprets them. In other words the word processor does not in itself contain the instructions

that perform the attack. Thus a word processor is not a software weapon (but the macro is).

Summary of Evaluation. The tools that challenged the definition the most were the traffic analyser and the port scanner. Both tools can very well be used by a system administrator for totally legitimate purposes. For example a traffic analyser can be used by an administrator to continuously monitor the traffic in a network and in that way detect anomalies signalling an intrusion. A port scanner can be used to test the security configuration of the system and especially the firewall set-up.

It is therefore important to remember that it is the code constituting the software that should contain instructions that are necessary and sufficient be used for a successful attack. If a port scanner does more than just scan for open ports in a firewall, it might very well perform actions successfully affecting the confidentiality of the scanned system and as a result be a software weapon, regardless of the context.

2.3 A Draft for a Taxonomy

The categories of the taxonomy are independent and the alternatives of each category together form a partition of the category. It is possible to use several alternatives (where applicable) in a category at the same time. In this way even combined software weapons can be unambiguously classified. This model, called *characteristics structure*, is suggested by Daniel Lough [12, p. 152].

In Table 1 the 15 categories and their alternatives are presented. The alternatives are then explained in separate paragraphs.

2.3.1 Type.

This category is used to distinguish an *atomic* software weapon from a *combined* and the alternatives therefore cannot be used together.

A combined software weapon is built of more than one stand-alone (atomic or combined) weapon. Such a weapon can utilise more than one alternative of a category. Usage of only one alternative from each category does not necessarily implicate an atomic weapon. In those circumstances this category indicates what type of weapon it is.

2.3.2 Affects.

At least one of the three elements *confidentiality*, *integrity* and *availability* has to be affected by a tool to make the tool a software weapon.

Table 1: The taxonomic categories and their alternatives

Category	Alternative 1	Alternative 2	Alternative 3
Type	atomic	combined	
Affects	confidentiality	integrity	availability
Duration of effect	temporary	permanent	
Targeting	manual	autonomous	
Attack	immediate	conditional	
Functional area	local	remote	
Sphere of operation	host-based	network-based	
Used vulnerability	CVE/CAN	other method	none
Topology	single source	distributed source	
Target of attack	single	multiple	
Platform dependency	dependent	independent	
Signature of code	monomorphic	polymorphic	
Signature of attack	monomorphic	polymorphic	
Signature when passive	visible	stealth	
Signature when active	visible	stealth	

These three elements together form the core of most of the definitions of IT security that exist today. Many of the schemes propose extensions to the core, but few of them abandon it completely.

2.3.3 Duration of effect.

This category states for how long the software weapon is affecting the attacked system. It is only the effect(s) the software weapon has on the system during the weapon's active phase that should be taken into account. If the effect of the software weapon ceases when the active phase is over, the duration of the effect is *temporary*, otherwise it is *permanent*.

Regarding an effect on the confidentiality of the attacked system, it can be temporary. If for example a software weapon e-mails confidential data to the attacker (or another unauthorised party), the duration of the effect is temporary. On the other hand, if the software weapon opens a back door into the system (and leaves it open), the effect is permanent.

2.3.4 Targeting.

The target of an attack can either be selected *manual*[ly] by the user, or *autonomous*[ly] (usually randomly) by the software weapon. Typical examples of autonom-

ously targeting software weapons are worms and viruses.

2.3.5 Attack.

The attack can be done *immediate*[ly] or *conditional*[ly]. If the timing of the attack is not governed by any conditions in the software, the software weapon uses immediate attack.

2.3.6 Functional Area.

If the weapon attacks its host computer, i.e. hardware directly connected to the processor running its instructions, it is a *local* weapon. If instead another physical entity is attacked, the weapon is *remote*.

The placement of the weapon on the host computer can be done either with the help of another, separate tool (including manual placement), or by the weapon itself. If the weapon establishes itself on the host computer (i.e. breaks the host computer's security) it certainly is local, but can still be remote at the same time. A weapon which is placed on the host computer manually (or by another tool) need not be local.

2.3.7 Sphere of Operation.

A weapon affecting network traffic in some way, for instance a traffic analyser, has a *network-based* operational area. A weapon affecting stationary data, for instance a weapon used to read password files, is *host-based*, even if the files are read over a network connection.

The definition of stationary data is data stored on a hard disk, in memory or on another type of physical storage media.

2.3.8 Used Vulnerability.

The alternatives of this category are *CVE/CAN*[13], *other method* and *none*. When a weapon uses a vulnerability or exposure [25] stated in the CVE, the CVE/CAN name of the vulnerability should be given[14] as the alternative (if several, give all of them).

[13]The term CAN (Candidate Number) indicates that the vulnerability or exposure is being investigated by the CVE Editorial Board for eventually receiving a CVE name [31].

[14]NIST (US National Institute of Standards and Technology) has initiated a meta-base called ICAT [32] based on the CVE list. This meta-base can be used to search for CVE/CAN names when classifying a software weapon.

The meta-base is described like this: 'ICAT is a fine-grained searchable index of standardized vulnerabilities that links users into publicly available vulnerability and patch information'. [33]

The alternative *other method* should be used with great discrimination and only if the flaw is not listed in the CVE, which then regularly must be checked to see if it has been updated with the new method. If so, the classification of the software weapon should be changed to the proper CVE/CAN name.

2.3.9 Topology.

An attack can be done from one *single source* or several concurrent *distributed sources*. In other words, the category defines the number of concurrent processes used for the attack. The processes should be mutually coordinated and running on separate and independent computers. If the computers are clustered or in another way connected as to make them simulate a single entity, they should be regarded as one.

2.3.10 Target of Attack.

This category is closely related to the category *topology* and has the alternatives *single* and *multiple*. As for the category topology, it is the number of involved entities that is important. A software weapon concurrently attacking several targets is consequently of the type multiple.

2.3.11 Platform Dependency.

The category states whether the software weapon (the executable code) can run on one or several platforms and the alternatives are consequently *dependent* and *independent*.

2.3.12 Signature of Code.

If a software weapon has functions for changing the signature of its code, it is *polymorphic*, otherwise it is *monomorphic*. The category should not be confused with *Signature when passive*.

2.3.13 Signature of Attack.

A software weapon can sometimes vary the way an attack is carried out, for example perform an attack of a specific type, but in different ways, or use different attacks depending on the status of the attacked system. For instance a dot-dot attack can be done either by using two dots, or by using the sequence `%2e%2e`. If the weapon has the ability to vary the attack, the type of attack is *polymorphic*, otherwise it is *monomorphic*.

2.3.14 Signature When Passive.

This category specifies whether the weapon is *visible* or uses any type of *stealth* when in a passive phase[15]. The stealth can for example be achieved by catching system interrupts, manipulating checksums or marking hard disk sectors as bad in the FAT (File Allocation Table).

2.3.15 Signature When Active.

A software weapon can be using instructions to provide *stealth* during its active phase. The stealth can be achieved in different ways, but the purpose is to conceal the effect and execution of the weapon. For example man-in-the-middle or spoofing weapons use stealth techniques in their active phases through simulating uninterrupted network connections.

If the weapon is not using any stealth techniques, the weapon is *visible*.

3 Examples

In this section, as a test, two software weapons are classified using the taxonomy. The weapons used are the distributed denial of service (DDoS) weapon Stacheldraht and the worm CodeRed. They were chosen for being well documented and well known.

The test is in no way exhaustive. It is only meant to function as a demonstration of what a classification can look like for a particular software weapon.

3.1 Stacheldraht.

The classification of the DDoS weapon Stacheldraht was made with the help of [34, 35] and looks like this:

Type: *combined*

Affects: *availability*

Duration of effect: *temporary*. The agents used to get the distributed characteristic of the weapon are installed permanently in the computers they reside on. To get them in place other tools are used [34], so the placing of the agents is to be considered as a separate attack not done with Stacheldraht.

The actual denial of service attack affects the attacked system until the attacker decides to quit.

[15] A passive phase is a part of the code constituting the software weapon where no functions performing an actual attack are executed.

Targeting: *manual*

Attack: *conditional*

Functional area: *remote*. As stated above, the placement of the agents is not considered an attack performed by Stacheldraht.

Sphere of operation: *network based*

Used vulnerability: *none*

Topology: *distributed source*

Target of attack: *multiple*

Platform dependency: *dependent*

Signature of code: *monomorphic*

Signature of attack: *monomorphic* The weapon can use ICMP flood, SYN flood, UDP flood, and Smurf style attacks, which are separate types of attacks.

Signature when passive: *visible*

Signature when active: *visible*, *stealth* The stealth is used in the communication between the different parts of the weapon (client, handler, and agent). This is done through using `ICMP_ECHOREPLY` packets and encrypted TCP [34].

3.2 CodeRed.

The classification of the worm CodeRed was made with the help of [36, 37, 38] and looks like this:

Type: *combined*

Affects: *integrity, availability*

Duration of effect: *temporary*. The documentation states that nothing is written to disk [37]. However, the weapon is also said to look for a file named 'NOTWORM' in the root of C:. How that file ends up there is not mentioned.

Regarding the defacing of the server it is done in memory by hooking and redirecting incoming request to the worm code during 10 hours [36], i.e. a temporary effect. The DoS attack is also limited in extent and therefore a temporary effect.

Targeting: *autonomous*

Attack: *conditional*

Functional area: *local, remote*. The weapon (in version 1) defaces the server it has infected and also performs a DoS attack on a specific IP address. Therefore it is both local and remote.

Sphere of operation: *host based, network based*. See the previous category.

Used vulnerability: *CVE-2001-0500 (idq.dll), CVE-2001-0506 (SSI)*

Topology: *single source*

Target of attack: *single, multiple*. The weapon executes a DoS attack on a single IP address. It is also divided into several $(99 + 1)$ threads, which all concurrently tries to infect (attack) randomly chosen IP addresses. This makes it both a single and multiple target attacking weapon.

Platform dependency: *dependent*

Signature of code: *monomorphic*

Signature of attack: *monomorphic* There are both a DoS attack and an infection mechanism, but each type of those two attacks are always executed in the same way.

Signature when passive: *visible*. The weapon is put to sleep when certain conditions are met. This cannot be regarded as using any stealth technique.

Signature when active: *visible*

4 Summary

The report has outlined a suggestion for a taxonomy, i.e. a classification scheme and a definition of software weapons. The definition part has been given much weight, because a classification scheme must have a solid base to work properly. To enable an unambiguous definition the emphasis was moved from the use of the weapon, to the technical (measurable) characteristics of the weapon. This gave the following formulation:

> A software weapon is software containing *instructions* that are necessary and sufficient for a *successful attack* on a *computer system*.

The classification part is meant to be used at a rather abstract level and for that reason the categories (and their alternatives) are chosen to be general properties held by all software weapons. A classification of a weapon must contain at least one alternative from each category.

By incorporating CVE names the taxonomy offers a connection to a global standard for naming vulnerabilities and exposures in software. This means that a meta-base of software weapons can be built, which can offer a global standardisation of the area.

The work done so far has been mainly theoretical. The next thing to do is to test the taxonomy empirically. Each category and its alternatives must be thoroughly tested to see if any of them needs to be changed.

Also the quality of the classification scheme needs to be tested. Software weapons related by common sense shall also have fairly similar classifications and unrelated weapons more or less be orthogonally classified.

References

[1] Richard Ford, *Malware*.
`http://www.malware.org/malware.htm`, accessed 17 July 2002.

[2] Marko Helenius, 'Problems, Advantages and Disadvantages of Malware Testing', in *EICAR 1999 Best Paper Proceedings*, 1999.
`http://conference.eicar.org/past_conferences/1999/other/Helenius.pdf`, accessed 18 July 2002.

[3] Jakub Kaminski and Hamish O'Dea, *How to smell a RAT - remote administration tools vs backdoor Trojans*.
`http://www.virusbtn.com/conference/this_year/abstracts/remote_administration.xml`, accessed 22 July 2002.
Only the abstract of the paper was available and therefore no references are made to the body of the document.

[4] A S Hornby, *Oxford advanced learner's dictionary of current English*, Oxford University Press, Oxford, 6 edition, 2000, ISBN 0-19-431510-X.

[5] Sun Tzu, *The Art of War*, 500 B.C.
`http://all.net/books/tzu/tzu.html`, accessed 12 June 2002.
Translation by Lionel Giles, 1910.

[6] Martin Karresand, 'TEBIT – Teknisk Beskrivningsmodell för IT-vapen (TEBIT. Technical characteristics' description model for IT-weapons)', Tech. Rep. FOI-R–0305–SE (Metodrapport/Methodology report), Command and Control Warfare Technology, FOI - Swedish Defence Research Agency, Linköping, Sweden, August 2001.

[7] Klaus Brunnstein, *From AntiVirus to AntiMalware Software and Beyond: Another Approach to the Protection of Customers from Dysfunctional System Behaviour*, Faculty for Informatics, University of Hamburg, Germany, July 1999.
`http://csrc.nist.gov/nissc/1999/proceeding/papers/p12.pdf`, accessed 22 July 2002.

[8] Marko Helenius, *A System to Support the Analysis of Antivirus Products' Virus Detection Capabilities*, PhD thesis, University of Tampere, Finland, 2002.
`http://acta.uta.fi/pdf/951-44-5394-8.pdf`, accessed 22 July 2002.

[9] Morton Swimmer, *Malware*.
`http://www.swimmer.org/morton/malware.html`, accessed 18 July 2002.

[10] Ian Whalley, Bill Arnold, David Chess, John Morar, Alla Segal, and Morton Swimmer, *An Environment for Controlled Worm Replication and Analysis or: Internet-inna-Box*, September 2000.
http://www.research.ibm.com/antivirus/SciPapers/VB2000INW.htm, accessed 18 July 2002.

[11] Ian Whalley, *Testing Times for Trojans*, October 1999.
http://www.research.ibm.com/antivirus/SciPapers/Whalley/inwVB99.html, accessed 18 July 2002.

[12] Daniel L Lough, *A Taxonomy of Computer Attacks with Applications to Wireless Networks*, PhD thesis, Virgina Polytechnic Institute and State University, April 2001.
http://scholar.lib.vt.edu/theses/available/etd-04252001-234145/unrestricted/lough.dissertation.pdf, accessed 13 June 2002.

[13] John D Howard, *An Analysis of Security Incidents on the Internet 1989-1995*, PhD thesis, Carnegie Mellon University, Pittsburg, April 1997.
http://www.cert.org/research/JHThesis/Word6/, accessed 12 June 2002.

[14] Ulf Lindqvist and Erland Jonsson, 'How to Systematically Classify Computer Security Intrusions', in *Proceedings of the 1997 IEEE Symposium on Security & Privacy*, Oakland, CA, 1997, pp. 154–163, IEEE Computer Society Press.
http://www.ce.chalmers.se/staff/ulfl/pubs/sp97ul.pdf, accessed 12 June 2002.

[15] Ivan V Krsul, *Software Vulnerability Analysis*, PhD thesis, Purdue University, May 1998.
http://www.acis.ufl.edu/~ivan/articles/main.pdf, accessed 13 June 2002.

[16] Robert J. Bagnall and Geoffrey French, *The Malware Rating System (MRS)^{TM}*, 2001.
http://www.dodccrp.org/6thICCRTS/Cd/Tracks/Papers/Track7/105_tr7.pdf, accessed 22 July 2002.

[17] Reto Haeni, *What is Information Warfare*, August 1996.
http://tangle.seas.gwu.edu/~reto/infowar/what.htm, accessed 27 June 2001.

[18] Winn Schwartau, *Information Warfare – Cyberterrorism: Protecting Your Personal Security in the Electronic Age*, Thunder's Mouth Press, New York, NY, 2 edition, 1996, ISBN 1-56025-132-8.

[19] Ivan Goldberg, January 2001.
`http://www.psycom.net/iwar.1.html`, accessed 26 June 2002.

[20] CERT (Computer Emergency Response Team), *CERT Advisory CA-1995-06 Security Administrator Tool for Analyzing Networks (SATAN)*, April 1995.
`http://www.cert.org/advisories/CA-1995-06.html`, accessed 12 June 2002.

[21] Sarah Gordon, *Devil's Advocate*, 1995.
`http://www.commandsoftware.com/virus/satan.html`, accessed 23 July 2002.

[22] CIAC (Computer Incidents Advisory Center), *Information Bulletin F-20: Security Administrator Tool for Analyzing Networks (SATAN)*, April 1995.
`http://www.ciac.org/ciac/bulletins/f-20.shtml`, accessed 12 June 2002.

[23] Dieter Gollmann, *Computer Security*, John Wiley & Sons, 1999, ISBN 0-471-97844-2.

[24] Carl E Landwehr, Alan R Bull, John P McDermott, and William S Choi, 'A Taxonomy of Computer Security Flaws', *ACM Computing Surveys*, vol. 26, no. 3, September 1994.
`http://chacs.nrl.navy.mil/publications/CHACS/1994/1994landwehr-acmcs.pdf`, accessed 12 June 2002.
A note taken from the text published on the web: 'As revised for publication in ACM Computing Surveys 26, 3 (Sept., 1994). This version, prepared for electronic distribution, reflects final revisions by the authors but does not incorporate Computing Surveys´ copy editing. It therefore resembles, but differs in minor details, from the published version. The figures, which have been redrawn for electronic distribution are slightly less precise, pagination differs, and Table 1 has been adjusted to reflect this'.

[25] CVE,
`http://cve.mitre.org/about/terminology.html`, accessed 4 July 2002.

[26] CVE,
`http://cve.mitre.org/about/index.html`, accessed 24 June 2002.

[27] MITRE, *The Early Years*.
`http://www.mitre.org/about/history.shtml`, accessed 12 June 2002.

[28] David Icove, Karl Seger, and William VonStorch, *Computer Crime: A Crimefighter's Handbook*, O´Reilley & Associates Inc, Sebastopol, CA, 1995.

[29] Anonymous, *Maximum Security – A Hacker's Guide to Protecting Your Internet Site and Network*, Sams Publishing, 2 edition, 1998, ISBN 0-672-31341-3.

[30] William Stallings, *Cryptography and Network Security, Principles and Practice*, Prentice Hall, 2 edition, 1999, ISBN 0-13-869017-0.

[31] CVE, `http://cve.mitre.org/docs/docs2000/naming_process.html`, accessed 12 June 2002.

[32] ICAT, `http://icat.nist.gov/icat.cfm`, accessed 12 June 2002.

[33] ICAT, `http://icat.nist.gov/icat_documentation.htm`, accessed 27 September 2002.

[34] David Dittrich, *The "stacheldraht" distributed denial of service attack tool*, December 1999. `http://staff.washington.edu/dittrich/misc/stacheldraht.analysis`, accessed 24 July 2002.

[35] David Dittrich, *The DoS Project's "trinoo" distributed denial of service attack tool*, October 1999. `http://staff.washington.edu/dittrich/misc/trinoo.analysis`, accessed 24 July 2002.

[36] eEye Digital Security, *.ida "Code Red" Worm*, July 2001. `http://www.eeye.com/html/Research/Advisories/AL20010717.html`, accessed 13 September 2002.

[37] Eric Chien, *CodeRed Worm*, Symantec, July 2002. `http://securityresponse.symantec.com/avcenter/venc/data/codered.worm.html`, accessed 18 July 2002.

[38] Trend Micro, *CODERED.A*, July 2001. `http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=CODERED.A&VSect=T`, accessed 24 July 2002.

Appendix B

Categorised Software Weapons

Categorised Software Weapons

Martin Karresand

1st April 2003

Contents

List of Tables

1 Introduction

In this appendix nine well known software weapons are categorised with the help of the TEBIT taxonomy and presented in separate sections. The chosen weapons are five distributed denial of service (DDoS) weapons and four worms[1].

First in each section the categorisation is shown using the two-dimensional view of the taxonomy. Some short comments are added to the categorisations where needed. The sources of the technical descriptions used for the categorisations are stated in the introduction to each weapon.

There is also a table showing the one-dimensional view of the taxonomy, which represents the basic data used in the calculation of the standard deviation in Section 5.4 in the main report. Each combination of category and alternative is presented as a 1 bit binary variable $s_i, i = 1, 2, \ldots, 34$. A complete categorisation forms a column vector.

2 The categorised weapons

The nine categorised weapons are presented in separate sections. Both a two-dimensional presentation, which is easier to read, and a one-dimensional view of each weapon is given in each section. The two views of a weapon shows the same categorisation, the only thing differing is the way the data is presented.

2.1 mstream

The classification of the DDoS weapon *mstream* was made with the help of [1]. The weapon is the most primitive of the DDoS weapons categorised here, because the analysed code was in an early stage of development. Most likely the creator will continue developing the weapon, to get it more potent. However, already in this beta stage the weapon is far from harmless.

The categorisation of the DDoS weapon looks like this:

Type: *combined*

Violates: *availability*

Duration of effect: *temporary*

Targeting: *manual*

Attack: *immediate*

[1] The term is defined in the old way; self-standing, replicating.

Functional area: *remote*

Affected data: *in transfer*

Used vulnerability: *none*

Topology of source: *distributed*

Target of attack: *multiple*

Platform dependency: *dependent*

Signature of replicated code: *not replicating*

Signature of attack: *monomorphic* The tool uses TCP ACK packets with forged, randomly generated source addresses.

Signature when passive: *visible*

Signature when active: *visible*

The one-dimensional categorisation is presented in Table 1.

2.2 Stacheldraht

The classification of the DDoS weapon Stacheldraht was made with the help of [2, 3]. The weapon is based on the source code of TFN (see Section 2.3) and has features similar to Trinoo's (see Section 2.5). It also uses encrypted communication between the attacker and the master program, as well as automated update of the daemons.

The categorisation of the DDoS weapon looks like this:

Type: *combined*

Violates: *availability*

Duration of effect: *temporary*. The agents used to get the distributed characteristic of the weapon are installed permanently in the computers they reside on. To get them in place other tools are used [2], so the placing of the agents is to be considered as a separate attack not done with Stacheldraht.

The actual denial of service attack affects the attacked system until the attacker decides to quit.

Targeting: *manual*

Attack: *conditional*

Functional area: *remote*. As stated above, the placement of the agents is not considered an attack performed by Stacheldraht.

Affected data: *network based*

Used vulnerability: *none*

Topology of source: *distributed*

Target of attack: *multiple*

Platform dependency: *dependent*

Signature of replicated code: *not replicating*

Signature of attack: *monomorphic* The weapon can use ICMP flood, SYN flood, UDP flood, and Smurf style attacks, which are separate types of attacks.

Signature when passive: *visible*

Signature when active: *visible, stealth* The stealth is used in the communication between the different parts of the weapon (client, handler, and agent). This is done through using `ICMP_ECHOREPLY` packets and encrypted TCP [2].

The one-dimensional categorisation is presented in Table 2.

2.3 TFN

The classification of the DDoS weapon *Tribe Flood Network (TFN)* was made with the help of [4]. This DDoS weapon uses ICMP packets for the communication between the different parts and thus is hard to detect (at least from looking at the packet flow). It was found when an installation of a Trinoo (see Section 2.5) network was stumbled upon.

The categorisation of the DDoS weapon looks like this:

Type: *combined*

Violates: *availability*

Duration of effect: *temporary*

Targeting: *manual*

Attack: *immediate*

Functional area: *remote*

Affected data: *in transfer*

Used vulnerability: *none*

Topology of source: *distributed*

Target of attack: *multiple*

Platform dependency: *dependent*

Signature of replicated code: *not replicating*

Signature of attack: *monomorphic* The tool can use UDP flood, ICMP flood, SYN flood, or Smurf style attacks, but which of the types to use is set by the user.

Signature when passive: *visible*

Signature when active: *visible, stealth* The tool uses `ICMP_ECHOREPLY` to provide stealth for the communication between the client and the daemon. It also sets the sequence number in the header of all packets to 0x0000 to imitate ping replies [4].

The one-dimensional categorisation is presented in Table 3.

2.4 TFN2K

The classification of the DDoS weapon *Tribe Flood Network 2000 (TFN2K)* was made with the help of [5]. By randomly chosing from three different methods of communication and from four different attack patterns, plus having the ability to spoof the source addresses, this weapon is making itself hard to find countermeasures for. By the way, it also encrypts the communication.

The categorisation of the weapon looks like this:

Type: *combined*

Violates: *availability*

Duration of effect: *temporary*

Targeting: *manual*

Attack: *immediate*

Functional area: *remote*

Affected data: *in transfer*

Used vulnerability: *none*

Topology of source: *distributed*

Target of attack: *multiple*

Platform dependency: *dependent*

Signature of replicated code: *not replicating*

Signature of attack: *polymorphic* The weapon has the ability to randomly vary the attack patterns between UDP flood, ICMP flood, SYN flood, or Smurf style attacks on its own.

Signature when passive: *visible*

Signature when active: *visible, stealth* The stealth is achieved by:

- encrypting the inter-tool communication
- inserting a random number of decoy packets in the communication flow
- randomly vary the type of packets used for the communication
- randomised packet headers
- completely silent daemons, they do not acknowledge the received commands
- having the daemons spawn a new child for each attack and tries to
- falsifying the child processes names on some platforms
- spoofing all packets between clients and daemons.

The one-dimensional categorisation is presented in Table 4.

2.5 Trinoo

The classification of the DDoS weapon *Trinoo* was made with the help of [3]. The Trinoo DDoS weapon was found together with the TFN (see Section 2.3) weapon when an intrusion of a system was investigated.

The categorisation of the DDoS weapon looks like this:

Type: *combined*

Violates: *availability*

Duration of effect: *temporary*

Targeting: *manual*

Attack: *immediate*

Functional area: *remote*

Affected data: *in transfer*

Used vulnerability: *none*

Topology of source: *distributed*

Target of attack: *multiple*

Platform dependency: *dependent*

Signature of replicated code: *not replicating*

Signature of attack: *monomorphic* The tool uses a UDP flood attack.

Signature when passive: *visible*

Signature when active: *visible*

The one-dimensional categorisation is presented in Table 5.

2.6 CodeRed

The classification of the worm CodeRed was made with the help of [6, 7, 8]. This is a famous worm, which pestered the users of the Internet in July 2001. It attacks servers running the Internet Information Services (IIS) and defaces them. At certain occasions the worm also makes a DoS attack on a specific IP address (the old IP address of the White House, Washington D.C.).

The categorisation of the worm looks like this:

Type: *combined*

Violates: *integrity;non-parasitic, availability*

Duration of effect: *temporary*. The documentation states that nothing is written to disk [6]. However, the weapon is also said to look for a file named 'NOT-WORM' in the root of C. How that file ends up there is not mentioned.

Regarding the defacing of the server it is done in memory by hooking and redirecting incoming request to the worm code during 10 hours [8], i.e. a temporary effect. The DoS attack is also limited in extent and therefore a temporary effect.

Targeting: *autonomous*

Attack: *conditional*

Functional area: *local, remote*. The weapon (in version 1) defaces the server it has infected and also performs a DoS attack on a specific IP address. Therefore it is both local and remote.

Affected data: *host based, network based*. See the previous category.

Used vulnerability: *CVE-2001-0500 (idq.dll), CVE-2001-0506 (SSI)*

Topology of source: *single*

Target of attack: *single, multiple*. The weapon executes a DoS attack on a single IP address. It is also divided into several $(99 + 1)$ threads, which all concurrently tries to infect (attack) randomly chosen IP addresses. This makes it both a single and multiple target attacking weapon.

Platform dependency: *dependent*

Signature of replicated code: *monomorphic*

Signature of attack: *monomorphic* There are both a DoS attack and an infection mechanism, but each type of those two attacks are always executed in the same way.

Signature when passive: *visible*. The weapon is put to sleep when certain conditions are met. This cannot be regarded as using any stealth technique.

Signature when active: *visible*

The one-dimensional categorisation is presented in Table 6.

2.7 Code Red II

The classification of the worm *Code Red II* was made with the help of [9, 10, 11]. CodeRed II utilises the same vulnerability in IIS servers as CodeRed (see Section 2.6), but the payload is different. This time a backdoor is installed on the attacked systems.

The categorisation of the worm looks like this:

Type: *combined*

Violates: *confidentiality*

Duration of effect: *temporary, permanent* The mechanism used to infect new victims is active for 24–48 hours. The installed backdoor is permanent.

Targeting: *autonomous*

Attack: *immediate*

Functional area: *local*

Affected data: *stationary*

Used vulnerability: *CVE-2001-0500 (idq.dll), CVE-2001-0506 (SSI)*

Topology of source: *single*

Target of attack: *multiple*

Platform dependency: *dependent*

Signature of replicated code: *monomorphic*

Signature of attack: *monomorphic*

Signature when passive: *visible*

Signature when active: *visible*

The one-dimensional categorisation is presented in Table 7.

2.8 Nimda

The classification of the worm *Nimda* was made with the help of [12, 13, 14]. This software weapon is really more than a worm. It spreads in four different ways; infection of files, via e-mail, via pages offered by IIS servers, and between shared partitions in local networks.

The categorisation of the software weapon looks like this:

Type: *combined*

Violates: *confidentiality, integrity;parasitic, integrity;non-parasitic* The weapon both attaches a file to itself to spread (integrity;parasitic), and changes web pages on infected IIS servers (integrity;non-parasitic).

Duration of effect: *permanent*

Targeting: *autonomous* The weapon uses randomly generated IP addresses as targets.

Attack: *immediate, conditional* The mass-mailing function is activated every 10th day, which makes it conditional. The other used types of attack are immediate.

Functional area: *local, remote* The remote part consists of attacks on file servers on the local network and computers running IIS on the Internet.

Affected data: *stationary*

Used vulnerability: *CVE-2000-0884, CVE-2001-0154*

Topology of source: *single*

Target of attack: *single* The mass mailing function is not to be regarded as an attack on multiple targets. Each mail contains a separate copy of the weapon.

Platform dependency: *dependent*

Signature of replicated code: *monomorphic*

Signature of attack: *monomorphic*

Signature when passive: *stealth*

Signature when active: *stealth*

The one-dimensional categorisation is presented in Table 8.

2.9 Sircam

The classification of the worm *Sircam* was made with the help of [15, 16, 17]. This is a trojan horse and a worm in the same packet. It has the ability to spread through Microsoft Windows network shares, as well as e-mail itself piggy-backing on a random document from the 'My Documents' folder in the infected computer.

The categorisation of the software weapon looks like this:

Type: *combined*

Violates: *confidentiality, integrity;parasitic, integrity;non-parasitic, availability* The weapon fills the remaining space on C: if a certain condition is met. It also merges itself with a randomly chosen file on the infected system and sends this resulting file on to infect new victims. Another used way of infecting new hosts is via network shares and in that case it copies itself to the share without any other file, thus it is both parasitic and non-parasitic.

Duration of effect: *permanent*

Targeting: *autonomous*

Attack: *immediate, conditional* The e-mailing part of the attack is of the immediate type.

Functional area: *local, remote* Apart from infecting files locally, the weapon also tries to infect all shares connected to the attacked computer, i.e. a remote functional area.

Affected data: *stationary*

Used vulnerability: *none*

Topology of source: *single*

Target of attack: *single*

Platform dependency: *dependent*

Signature of replicated code: *monomorphic*

Signature of attack: *monomorphic*

Signature when passive: *stealth*

Signature when active: *stealth*

The one-dimensional categorisation is presented in Table 9.

Table 1: The one-dimensional categorisation of *mstream*

Category	Alternative	0/1
Type	atomic	0
Type	combined	1
Violates	confidentiality	0
Violates	integrity;parasitic	0
Violates	integrity;non-parasitic	0
Violates	availability	1
Dur. of effect	temporary	1
Dur. of effect	permanent	0
Targeting	manual	1
Targeting	autonomous	0
Attack	immediate	1
Attack	conditional	0
Funct. area	local	0
Funct. area	remote	1
Affected data	stationary	0
Affected data	in transfer	1
Used vuln.	CVE/CAN	0
Used vuln.	other vuln.	0
Used vuln.	none	1
Topol. of source	single	0
Topol. of source	distributed	1
Target of attack	single	0
Target of attack	multiple	1
Platform depend.	dependent	1
Platform depend.	independent	0
Sign. of repl. code	monomorphic	0
Sign. of repl. code	polymorphic	0
Sign. of repl. code	not replicating	1
Sign. of attack	monomorphic	1
Sign. of attack	polymorphic	0
Sign. when passive	visible	1
Sign. when passive	stealth	0
Sign. when active	visible	1
Sign. when active	stealth	0

Table 2: The one-dimensional categorisation of *Stacheldraht*

Category	Alternative	0/1
Type	atomic	0
Type	combined	1
Violates	confidentiality	0
Violates	integrity;parasitic	0
Violates	integrity;non-parasitic	0
Violates	availability	1
Dur. of effect	temporary	1
Dur. of effect	permanent	0
Targeting	manual	1
Targeting	autonomous	0
Attack	immediate	0
Attack	conditional	1
Funct. area	local	0
Funct. area	remote	1
Affected data	stationary	0
Affected data	in transfer	1
Used vuln.	CVE/CAN	0
Used vuln.	other vuln.	0
Used vuln.	none	1
Topol. of source	single	0
Topol. of source	distributed	1
Target of attack	single	0
Target of attack	multiple	1
Platform depend.	dependent	1
Platform depend.	independent	0
Sign. of repl. code	monomorphic	0
Sign. of repl. code	polymorphic	0
Sign. of repl. code	not replicating	1
Sign. of attack	monomorphic	1
Sign. of attack	polymorphic	0
Sign. when passive	visible	1
Sign. when passive	stealth	0
Sign. when active	visible	1
Sign. when active	stealth	1

Table 3: The one-dimensional categorisation of *TFN*

Category	Alternative	0/1
Type	atomic	0
Type	combined	1
Violates	confidentiality	0
Violates	integrity;parasitic	0
Violates	integrity;non-parasitic	0
Violates	availability	1
Dur. of effect	temporary	1
Dur. of effect	permanent	0
Targeting	manual	1
Targeting	autonomous	0
Attack	immediate	1
Attack	conditional	0
Funct. area	local	0
Funct. area	remote	1
Affected data	stationary	0
Affected data	in transfer	1
Used vuln.	CVE/CAN	0
Used vuln.	other vuln.	0
Used vuln.	none	1
Topol. of source	single	0
Topol. of source	distributed	1
Target of attack	single	0
Target of attack	multiple	1
Platform depend.	dependent	1
Platform depend.	independent	0
Sign. of repl. code	monomorphic	0
Sign. of repl. code	polymorphic	0
Sign. of repl. code	not replicating	1
Sign. of attack	monomorphic	1
Sign. of attack	polymorphic	0
Sign. when passive	visible	1
Sign. when passive	stealth	0
Sign. when active	visible	1
Sign. when active	stealth	1

Table 4: The one-dimensional categorisation of *TFN2K*

Category	Alternative	0/1
Type	atomic	0
Type	combined	1
Violates	confidentiality	0
Violates	integrity;parasitic	0
Violates	integrity;non-parasitic	0
Violates	availability	1
Dur. of effect	temporary	1
Dur. of effect	permanent	0
Targeting	manual	1
Targeting	autonomous	0
Attack	immediate	1
Attack	conditional	0
Funct. area	local	0
Funct. area	remote	1
Affected data	stationary	0
Affected data	in transfer	1
Used vuln.	CVE/CAN	0
Used vuln.	other vuln.	0
Used vuln.	none	1
Topol. of source	single	0
Topol. of source	distributed	1
Target of attack	single	0
Target of attack	multiple	1
Platform depend.	dependent	1
Platform depend.	independent	0
Sign. of repl. code	monomorphic	0
Sign. of repl. code	polymorphic	0
Sign. of repl. code	not replicating	1
Sign. of attack	monomorphic	0
Sign. of attack	polymorphic	1
Sign. when passive	visible	1
Sign. when passive	stealth	0
Sign. when active	visible	1
Sign. when active	stealth	1

Table 5: The one-dimensional categorisation of *Trinoo*

Category	Alternative	0/1
Type	atomic	0
Type	combined	1
Violates	confidentiality	0
Violates	integrity;parasitic	0
Violates	integrity;non-parasitic	0
Violates	availability	1
Dur. of effect	temporary	1
Dur. of effect	permanent	0
Targeting	manual	1
Targeting	autonomous	0
Attack	immediate	1
Attack	conditional	0
Funct. area	local	0
Funct. area	remote	1
Affected data	stationary	0
Affected data	in transfer	1
Used vuln.	CVE/CAN	0
Used vuln.	other vuln.	0
Used vuln.	none	1
Topol. of source	single	0
Topol. of source	distributed	1
Target of attack	single	0
Target of attack	multiple	1
Platform depend.	dependent	1
Platform depend.	independent	0
Sign. of repl. code	monomorphic	0
Sign. of repl. code	polymorphic	0
Sign. of repl. code	not replicating	1
Sign. of attack	monomorphic	1
Sign. of attack	polymorphic	0
Sign. when passive	visible	1
Sign. when passive	stealth	0
Sign. when active	visible	1
Sign. when active	stealth	0

Table 6: The one-dimensional categorisation of *CodeRed*

Category	Alternative	0/1
Type	atomic	0
Type	combined	1
Violates	confidentiality	0
Violates	integrity;parasitic	0
Violates	integrity;non-parasitic	1
Violates	availability	1
Dur. of effect	temporary	1
Dur. of effect	permanent	0
Targeting	manual	0
Targeting	autonomous	1
Attack	immediate	0
Attack	conditional	1
Funct. area	local	1
Funct. area	remote	1
Affected data	stationary	1
Affected data	in transfer	1
Used vuln.	CVE/CAN	1
Used vuln.	other vuln.	0
Used vuln.	none	0
Topol. of source	single	1
Topol. of source	distributed	0
Target of attack	single	1
Target of attack	multiple	1
Platform depend.	dependent	1
Platform depend.	independent	0
Sign. of repl. code	monomorphic	1
Sign. of repl. code	polymorphic	0
Sign. of repl. code	not replicating	0
Sign. of attack	monomorphic	1
Sign. of attack	polymorphic	0
Sign. when passive	visible	1
Sign. when passive	stealth	0
Sign. when active	visible	1
Sign. when active	stealth	0

Table 7: The one-dimensional categorisation of *CodeRed II*

Category	Alternative	0/1
Type	atomic	0
Type	combined	1
Violates	confidentiality	1
Violates	integrity;parasitic	1
Violates	integrity;non-parasitic	0
Violates	availability	0
Dur. of effect	temporary	1
Dur. of effect	permanent	1
Targeting	manual	0
Targeting	autonomous	1
Attack	immediate	1
Attack	conditional	0
Funct. area	local	1
Funct. area	remote	0
Affected data	stationary	1
Affected data	in transfer	0
Used vuln.	CVE/CAN	1
Used vuln.	other vuln.	0
Used vuln.	none	0
Topol. of source	single	1
Topol. of source	distributed	0
Target of attack	single	0
Target of attack	multiple	1
Platform depend.	dependent	1
Platform depend.	independent	0
Sign. of repl. code	monomorphic	1
Sign. of repl. code	polymorphic	0
Sign. of repl. code	not replicating	0
Sign. of attack	monomorphic	1
Sign. of attack	polymorphic	0
Sign. when passive	visible	1
Sign. when passive	stealth	0
Sign. when active	visible	1
Sign. when active	stealth	0

Table 8: The one-dimensional categorisation of *Nimda*

Category	Alternative	0/1
Type	atomic	0
Type	combined	1
Violates	confidentiality	1
Violates	integrity;parasitic	1
Violates	integrity;non-parasitic	1
Violates	availability	0
Dur. of effect	temporary	0
Dur. of effect	permanent	1
Targeting	manual	0
Targeting	autonomous	1
Attack	immediate	1
Attack	conditional	1
Funct. area	local	1
Funct. area	remote	1
Affected data	stationary	1
Affected data	in transfer	0
Used vuln.	CVE/CAN	1
Used vuln.	other vuln.	0
Used vuln.	none	0
Topol. of source	single	1
Topol. of source	distributed	0
Target of attack	single	1
Target of attack	multiple	0
Platform depend.	dependent	1
Platform depend.	independent	0
Sign. of repl. code	monomorphic	1
Sign. of repl. code	polymorphic	0
Sign. of repl. code	not replicating	0
Sign. of attack	monomorphic	1
Sign. of attack	polymorphic	0
Sign. when passive	visible	0
Sign. when passive	stealth	1
Sign. when active	visible	0
Sign. when active	stealth	1

Table 9: The one-dimensional categorisation of *Sircam*

Category	Alternative	0/1
Type	atomic	0
Type	combined	1
Violates	confidentiality	1
Violates	integrity;parasitic	1
Violates	integrity;non-parasitic	1
Violates	availability	1
Dur. of effect	temporary	0
Dur. of effect	permanent	1
Targeting	manual	0
Targeting	autonomous	1
Attack	immediate	1
Attack	conditional	1
Funct. area	local	1
Funct. area	remote	1
Affected data	stationary	1
Affected data	in transfer	0
Used vuln.	CVE/CAN	0
Used vuln.	other vuln.	0
Used vuln.	none	1
Topol. of source	single	1
Topol. of source	distributed	0
Target of attack	single	1
Target of attack	multiple	0
Platform depend.	dependent	1
Platform depend.	independent	0
Sign. of repl. code	monomorphic	1
Sign. of repl. code	polymorphic	0
Sign. of repl. code	not replicating	0
Sign. of attack	monomorphic	1
Sign. of attack	polymorphic	0
Sign. when passive	visible	0
Sign. when passive	stealth	1
Sign. when active	visible	0
Sign. when active	stealth	1

References

[1] David Dittrich, George Weaver, Sven Dietrich, and Neil Long, *The "mstream" distributed denial of service attack tool*, May 2000. `http://staff.washington.edu/dittrich/misc/mstream.analysis.txt`, accessed 16 November 2002.

[2] David Dittrich, *The "stacheldraht" distributed denial of service attack tool*, December 1999. `http://staff.washington.edu/dittrich/misc/stacheldraht.analysis`, accessed 24 July 2002.

[3] David Dittrich, *The DoS Project's "trinoo" distributed denial of service attack tool*, October 1999. `http://staff.washington.edu/dittrich/misc/trinoo.analysis.txt`, accessed 24 July 2002.

[4] David Dittrich, *The "Tribe Flood Network" distributed denial of service attack tool*, October 1999. `http://staff.washington.edu/dittrich/misc/tfn.analysis.txt`, accessed 16 November 2002.

[5] Jason Barlow and Woody Thrower, *TFN2K – An Analysis*, March 2000. `http://packetstormsecurity.nl/distributed/TFN2k_Analysis-1.3.txt`, accessed 17 November 2002.

[6] Eric Chien, *CodeRed Worm*, Symantec, July 2002. `http://securityresponse.symantec.com/avcenter/venc/data/codered.worm.html`, accessed 24 July 2002.

[7] Trend Micro, *CODERED.A*, July 2001. `http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=CODERED.A&VSect=T`, accessed 24 July 2002.

[8] eEye Digital Security, *.ida "Code Red" Worm*, July 2001. `http://www.eeye.com/html/Research/Advisories/AL20010717.html`, accessed 13 September 2002.

[9] Eric Chien and Peter Szor, *CodeRed II*, Symantec, July 2002. `http://securityresponse.symantec.com/avcenter/venc/data/codered.ii.html`, accessed 18 October 2002.

[10] Trend Micro, *CODERED.C*, August 2001. http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=CODERED.C&VSect=T, accessed 18 October 2002.

[11] eEye Digital Security, *CodeRedII Worm Analysis*, August 2001. http://www.eeye.com/html/Research/Advisories/AL20010804.html, accessed 18 October 2002.

[12] Eric Chien, *W32.Nimda.A@mm*, Symantec, July 2002. http://securityresponse.symantec.com/avcenter/venc/data/w32.nimda.a@mm.html#technicaldetails, accessed 21 October 2002.

[13] Trend Micro, *PE_NIMDA.A*, October 2001. http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?Vname=PE_NIMDA.A&VSect=T, accessed 21 October 2002.

[14] K Tocheva, G Erdelyi, A Podrezov, S Rautiainen, and M Hypponen, *Nimda*, F-Secure, September 2001. http://www.europe.f-secure.com/v-descs/nimda.shtml, accessed 21 October 2002.

[15] Peter Ferrie and Peter Szor, *W32.Sircam.Worm@mm*, Symantec, July 2002. http://securityresponse.symantec.com/avcenter/venc/data/w32.sircam.worm@mm.html#technicaldetails, accessed 23 October 2002.

[16] Gergely Erdelyi and Alexey Podrezov, *Sircam*, F-Secure, July 2001. http://www.europe.f-secure.com/v-descs/sircam.shtml, accessed 23 October 2002.

[17] Trend Micro, *WORM_SIRCAM.A*, October 2001. http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=WORM_SIRCAM.A&VSect=T, accessed 23 October 2002.

Appendix C

Redefined Terms

Redefined Terms

Martin Karresand

1st April 2003

1 Introduction

The definitions of the three terms *trojan horse*, *virus*, and *worm* presented in this appendix are really only proposals. They have been made using the main characteristics of the terms, extracted from some of the more widespread definitions used today. However, to be usable the defintions of the terms need to be unanimously agreed upon by the whole research community.

2 Redefined terms

Each definition is presented as a table in a separate section, where also a short explanation of the underlying ideas are included.

The requirement to use at least one alternative from each category is still valid regarding the software weapons belonging to a class, but it is dropped when using a wildcard in a categorisation of a definition. Setting a combination of a category and alternative to 0 is *not* the same as using an alternative in that category.

2.1 Trojan horse

A trojan horse is often defined as a program performing destructive, or at least not by the user authorised functions, when at the same time posing as a legitimate program.

Another definition with the same meaning as the one above can be formulated as the weapon needing a user to start the execution by tricking him or her into it. This way of defining the term is the preferred one and hence the definition shown in Table 1. As can be seen the old definition really still fits the new one.

2.2 Virus

A virus is a program which replicates and also is parasitic. Often it is regarded as only attacking the host computer, the computer it is residing on, but this definition is far from being generally accepted.

The new proposed definition, which can be seen in Table 2, is meant to be rather general. There are several categories which are marked with *wildcard* that might be used to narrow the definition a bit. For example the targeting of a virus might be set to 1, because they tend to spread uncontrollably. It is also possible to set the *Functional area; local* to 1, because of the virus being regarded as always replicating on its own host. To separate the viruses from trojan horses the *Used vulnerability; none* might be set to 0. Mostly a virus have a single source topology, at least a simple virus, and consequently that alternative might be set to 1.

2.3 Worm

The worms are often regarded as forming a sub-class of viruses, but not always. Both a virus and a worm are defined as replicating. The things setting them apart are that worms are defined as being self-standing (not being parasitic) and often also as replicating over network connections.

The new, proposed definition is shown in Table 3 and as for the new definition of virus it is rather general and hence might also be narrowed down a bit.

First of all, the *Affects; integrity;non-parasitic* alternative may really be needed, because the worm installs itself on the system and thus affects the integrity of the system, although not in a parasitic way. It might also be necessary to separate a worm from a trojan horse, and therefore the *Used vulnerability; none* may be set to 0. Also the targeting is mostly autonomous, but not necessarily. Because a worm by some is regarded to only replicate over network connections, the alternative *Functional area; remote* might be set to 1 and *Functional area; local* to 0.

Table 1: The redefined term *Trojan horse*

Category	Alternative	0/1/wildcard
Type	atomic	wildcard
Type	combined	wildcard
Violates	confidentiality	wildcard
Violates	integrity;parasitic	wildcard
Violates	integrity;non-parasitic	wildcard
Violates	availability	wildcard
Dur. of effect	temporary	wildcard
Dur. of effect	permanent	wildcard
Targeting	manual	wildcard
Targeting	autonomous	wildcard
Attack	immediate	wildcard
Attack	conditional	wildcard
Funct. area	local	wildcard
Funct. area	remote	wildcard
Affected data	stationary	wildcard
Affected data	in transfer	wildcard
Used vuln.	CVE/CAN	0
Used vuln.	other vuln.	0
Used vuln.	none	1
Topol. of source	single	wildcard
Topol. of source	distributed	wildcard
Target of attack	single	wildcard
Target of attack	multiple	wildcard
Platform depend.	dependent	wildcard
Platform depend.	independent	wildcard
Sign. of repl. code	monomorphic	wildcard
Sign. of repl. code	polymorphic	wildcard
Sign. of repl. code	not replicating	wildcard
Sign. of attack	monomorphic	wildcard
Sign. of attack	polymorphic	wildcard
Sign. when passive	visible	wildcard
Sign. when passive	stealth	wildcard
Sign. when active	visible	wildcard
Sign. when active	stealth	wildcard

Table 2: The redefined term *virus*

Category	Alternative	0/1/wildcard
Type	atomic	wildcard
Type	combined	wildcard
Violates	confidentiality	wildcard
Violates	integrity;parasitic	1
Violates	integrity;non-parasitic	wildcard
Violates	availability	wildcard
Dur. of effect	temporary	wildcard
Dur. of effect	permanent	wildcard
Targeting	manual	wildcard
Targeting	autonomous	wildcard
Attack	immediate	wildcard
Attack	conditional	wildcard
Funct. area	local	wildcard
Funct. area	remote	wildcard
Affected data	stationary	wildcard
Affected data	in transfer	wildcard
Used vuln.	CVE/CAN	wildcard
Used vuln.	other vuln.	wildcard
Used vuln.	none	wildcard
Topol. of source	single	wildcard
Topol. of source	distributed	wildcard
Target of attack	single	wildcard
Target of attack	multiple	wildcard
Platform depend.	dependent	wildcard
Platform depend.	independent	wildcard
Sign. of repl. code	monomorphic	wildcard
Sign. of repl. code	polymorphic	wildcard
Sign. of repl. code	not replicating	0
Sign. of attack	monomorphic	wildcard
Sign. of attack	polymorphic	wildcard
Sign. when passive	visible	wildcard
Sign. when passive	stealth	wildcard
Sign. when active	visible	wildcard
Sign. when active	stealth	wildcard

Table 3: The redefined term *worm*

Category	Alternative	0/1/wildcard
Type	atomic	wildcard
Type	combined	wildcard
Violates	confidentiality	wildcard
Violates	integrity;parasitic	0
Violates	integrity;non-parasitic	wildcard
Violates	availability	wildcard
Dur. of effect	temporary	wildcard
Dur. of effect	permanent	wildcard
Targeting	manual	wildcard
Targeting	autonomous	wildcard
Attack	immediate	wildcard
Attack	conditional	wildcard
Funct. area	local	wildcard
Funct. area	remote	wildcard
Affected data	stationary	wildcard
Affected data	in transfer	wildcard
Used vuln.	CVE/CAN	wildcard
Used vuln.	other vuln.	wildcard
Used vuln.	none	wildcard
Topol. of source	single	wildcard
Topol. of source	distributed	wildcard
Target of attack	single	wildcard
Target of attack	multiple	wildcard
Platform depend.	dependent	wildcard
Platform depend.	independent	wildcard
Sign. of repl. code	monomorphic	wildcard
Sign. of repl. code	polymorphic	wildcard
Sign. of repl. code	not replicating	0
Sign. of attack	monomorphic	wildcard
Sign. of attack	polymorphic	wildcard
Sign. when passive	visible	wildcard
Sign. when passive	stealth	wildcard
Sign. when active	visible	wildcard
Sign. when active	stealth	wildcard