Current Status of the CARO Malware Naming Scheme

Dr. Vesselin Bontchev, anti–virus researcher
FRISK Software International
Thverholt 18, IS–105 Reykjavik, Iceland
E–mail: bontchev@complex.is

***Abstract**: The CARO malware naming scheme was created more than 15 years ago. To this date, it remains the naming scheme that is the most widely used in anti–virus products—despite being criticized left and right and the fact that no product has absolute compliance with it. One frequent criticism is that detailed documentation of the up–to–date status of the Scheme is difficult to find and that this hampers the Scheme's popularity. This paper attempts to solve this problem. It documents the CARO malware naming scheme completely, including the recently introduced changes. It will be made freely available on the Web and will be continuously updated as new changes are introduced. Its purpose is to serve as an easily and publicly accessible documentation of the latest state of the CARO Malware Naming Scheme.*

1. Introduction

In the first part of this paper we shall discuss why a malware naming scheme is needed, the various unsuccessful malware naming schemes that have been tried in the past and the history of the CARO Malware Naming Scheme.

1.1. The Malware Naming Mess

Practically since computer viruses first appeared, there has been a huge disparity in the names used by the different anti–virus products for one and the same malware program. And since about that time users have been complaining about it, as it often leads to serious confusion.

First, it can lead to failure of two people discussing a malware program to understand each other unless both of them have a copy of the same program and know that it is indeed one and the same malware program. For instance, if some user's scanner detects a virus and names it `MyDoom.D`, it is natural for the user to ask "What does this virus do?." Unfortunately, different scanners (or different versions of one and the same scanner or even different virus definitions database updates of one and the same scanner) can call different viruses by this name. Therefore, the anti–virus experts are unable to help the user unless they receive a sample of the virus so that they can examine it. However, in several cases this is not possible or not allowed, for instance, if the user asks in some kind of public forum where swapping of virus samples is discouraged, or if the sample contains confidential information that the user is not willing to disclose (e.g., a Microsoft Office document infected with a macro virus). In other cases, it may take a lot of time until a sample is received by the anti–virus expert and the user often needs an immediate answer, for instance when, due to the infection, the company's machines are down and further action would depend on what kind of damage this particular malware program might have done.

Second, many users use more than one scanner. The reason for this is that no scanner is able to detect all known malware programs, but the different scanners usually do not detect exactly the same subset of the known malware. Therefore, by using more than one product, the users hope to increase their chances of detecting the particular malware that might be attacking their machine(s). Unfortunately, if the different products use different names for the same malware programs, this can lead to serious confusion. In one particular case, a user reported that she used several different scanners and they discovered many different viruses on her machine—namely, the viruses `Kampana`, `Telefonica`, `Telecom`, `Antitel`, and `Spanish`. She used one of the products to remove one of the viruses and the other products suddenly stopped reporting the other viruses. From this, she concluded that the other viruses had noticed the death of their friend and were now hiding. ☺ In reality, of course, these are five different names for one and the same virus, used by different virus–specific programs.

Let us examine the various reasons why the malware naming mess exists ([Bontchev04]).

1.1.1. Glut

Although significant virus naming confusion existed even in the early days when there were only a few dozen computer viruses in existence, nowadays one of the main causes for this confusion is the ever–increasing speed at which new malware programs appear and spread (or are distributed, in the case of non–viral malware). The number of known malware programs nowadays probably approaches 200,000. Even worse, nobody is capable of saying exactly how many such malware programs exist without making an error of in the order of tens of thousands. Often more than 5,000 new malware programs appear monthly. Some of them cause real pandemics, spreading all over the world in a matter of hours.

1.1.2. Lack of Time and Other Resources

The above reason leads us to the next one. Nowadays the anti–virus people are simply physically incapable of examining every single new malware program that appears and agreeing on a common, reasonable name for it, nor is it physically possible for anyone to go over all the existing malware and deciding exactly what the name of each malware program should be. We are far too busy preparing speedy detection (and only sometimes disinfection) of the new malware that constantly appears and rushing it out the door to meet user demand.

Even worse, the people in our virus labs who are often the ones who implement detection of the new malware are competent enough to implement such detection (over time, we have made adding detection of new malware to our products a reasonably fast and simple task, so it is easy to train people to do it) but they usually do not have the much higher level of competence required to properly classify and name a new malware program. The latter requires careful analysis of the malware and detailed knowledge of thousands of other such malware programs, in order to classify and name the new one correctly.

Finally, once we have chosen a name for a new malware program, it is often too expensive to change it later if it is found to be "incorrect". The original name might have already gone into press releases, on–line virus descriptions and even printed documentation. Changing it would cost both time, money and effort and would be likely to lead to user confusion.

1.1.3. Lack of a Common Virus Naming Standard

In order to have the different anti–virus products name malware a uniform manner, a common virus naming standard is needed. It has to have several properties in order to be good enough and acceptable to everyone.

First of all, it has to make sense. Similar malware programs should have similar names. Of course, how to define malware similarity is a completely different question—it could be similarity in contents, similarity in behavior, similarity in other properties or even similarity in time and/or place of discovery.

Second, it has to be understandable and usable by the general user. The user can remember and report on the phone a name like "`Melissa`" but would have trouble even pronouncing a name like "`BfPsHgTx9435`".

Third, the standard should be usable by all (or at least by the vast majority) of anti–virus products. This is not as easy to achieve as it might seem at first glance. Contrary to popular belief, not all anti–virus products work by just looking for a bunch of scan strings in all files. In fact, even products that look similarly to the user often use wildly different technology for malware detection and have very different internal limitations. For instance, the different scanners have different limits on the maximal length of a malware name. For some scanners (e.g., F–PROT) this maximal length cannot be determined in advance and depends on the contents of the name. (The names are stored in Huffman–compressed form and the limit is on the size of the compressed name, so the maximal length of the name the scanner can use depends on how well this name compresses.) Other scanners can report malware names only in upper case. Yet others have various limitations on what characters they can use in a malware name. And so on and so forth. The various conditions, limitations and dependencies are just too numerous to list here.

From the beginning, the CARO Malware Naming Scheme has striven to fulfill this need. Unfortunately, CARO is not an official standard–setting body, so we are unable to establish a standard. The best we could hope for was to come up with a Naming Scheme that was good enough to be acceptable to the vast majority of anti–virus producers and present it as a set of guidelines, hoping that these guidelines would be widely adopted by the industry.

It is our belief that we have managed to achieve this goal. The reason why the malware naming mess still exists is because we have not solved the other problems listed in this section, but solving them has never been our goal and solving some of them is obviously beyond our capability.

1.1.4. Lack of Reliable Means for Automatic Malware Identification

A good malware naming scheme specifies how the name of a new piece of malware should be constructed. It does not, however, specify what the name of a particular piece of malware should be.

For instance, both "`Win32/MyDoom.BQ`" and "`Win32/MyDoom.ED`" are valid virus names in the context of the CARO Malware Naming Scheme. But if one scanner uses the first name for a particular virus and another scanner uses the second one for the same virus, they are both compliant with the naming scheme yet we already have a naming confusion. The naming scheme cannot tell us which of the two scanners is right and which is wrong. In fact, in certain circumstances they can both be wrong yet still both be compliant with the naming scheme.

In order to resolve this problem, we need reliable means of mapping any given malware program with its correct name, chosen in compliance with the naming scheme. Unfortunately, in the general case, we do not have such means.

In some particular cases this problem *has* been resolved to a satisfactory degree. For instance, in the case of macro malware, we have a tool (F–VBACRC) that produces various kinds of identification data for a given macro malware program. The data is produced by plug–ins supplied by several anti–virus producers (Computer Associates, FRISK Software, McAfee, Symantec and others) and is sufficient for the respective producer to implement detection, recognition, identification and removal of the respective malware even without having a sample of it. Although neither set of data identifies a macro malware program with 100% exactness (especially true in the case of polymorphic viruses), the combined set of outputs from all the plug–ins is usually sufficient to identify a particular macro malware program. Furthermore, as new macro malware is discovered, its identification data is posted to a specialized mailing list and kept in various repositories from where it can be extracted and examined later, when a need arises to determine whether a particular macro malware program is a known one or not.

Unfortunately, we simply do not have an equivalent tool for the other kinds of malware. (In fact, even in the macro malware field, there are some kinds of macro malware—or macro malware supporting platforms, like Visio or Microsoft Access—that the tool does not support and/or identify reliably.)

We do have somewhat similar tools for other classes of malware—but they are vastly inferior and don't cover all classes of malware anyway. For instance, we have SCRID, a tool written by Dmitry Gryaznov from McAfee, which produces identification data for script malware. Unfortunately, this tool does not support plug–ins (i.e., it produces identification data that makes sense only to one particular anti–virus producer) and it cannot sufficiently handle polymorphic script viruses or parasitic script viruses.

We also have another tool, `PEInfo`, that outputs various useful information about the structure and contents of a given PE–EXE file. This is often useful for identifying self–contained malware that resides in such files. Unfortunately, it cannot handle parasitic and/or polymorphic viruses. Furthermore, we don't have a repository where we routinely announce the identification data of every new malware program that the tool can handle.

With the lack of proper tools, the only way to solve the mapping problem would be to have a "reference" malware collection, a high–quality malware collection that contains every known malware program and that is organized in a way that would permit easy determination of what the correct name of any given malware program is. Unfortunately, given the huge number of malware in existence, it is simply humanly impossible for any single person to maintain such a collection. What is needed is a reasonably large team of highly competent anti–virus researchers, employed full–time in the maintenance of such a collection.

Unfortunately, for various reasons this is not achievable, either. Virtually every anti–virus researcher in the world who already has the level of competence required for such a job is already employed by some anti–virus company. The maintenance of such a reference malware collection has do be done by completely independent people, in order to avoid various kinds of bias. But even without this requirement,

the anti–virus researchers with the required level of competence are already overloaded with everyday work in the companies that employ them and they simply cannot afford to spend the additional time and effort necessary to properly maintain a reference malware collection.

Finally, besides the competence–related issues, there are the issues of trust. Who should be allowed access to the reference malware collection? Answering this question is not as easy as it might seem at first glance. It would be unwise to allow full access to a huge collection of malware to just about anyone who happens to make the claim of being an anti–virus researcher. Doing so is a sure–fire way for the collection to end up on the various virus exchange (VX) sites. On the other hand, how does one ensure that a legitimate newcomer to the anti–virus industry *does* get access to the collection?

If the maintainers of the collection only accept new malware and return only the information on what the name of a particular submitted malware program should be, this puts the person who submits a new malware program, but does not have access to the collection at a significant disadvantage. They will have to wait until the collection maintainers tell them what the name of the new malware should be (a process which could take considerable time, especially if the submitted malware program is indeed new and needs to be analyzed first) while at the same time those who are considered trustworthy and have access to the collection (and who are competitors) will have the unfair advantage of getting the new malware and implementing detection of it while the original discoverer is still waiting.

In order to resolve *this* problem, we need an identification tool that can provide identification data for any given malware (so that this data and not the malware itself can be made available to anyone who needs the information) *and* automated tools must exist for properly classifying and naming newly submitted malware (see also the next subsection). Unfortunately, as already mentioned above, such tools simply do not exist, so we end up with a Catch 22 situation.☹

1.1.5. Lack of Reliable Means for Automatic Malware Classification

When new malware is discovered, it is not sufficient to determine that it is, indeed, *new* (which is already difficult enough, as explained in the previous subsection). It is also not enough to have a naming scheme that tells us *how* the name of the new malware should be constructed. We also need to determine *what* exactly the new name should be.

In a naming scheme based on classification and similarity (and all naming schemes essentially are of this kind), we need tools to help us determine which known malware is “most similar” to the new one—so that we can pick a name for the new malware that is similar to that “most similar” known malware.

Depending on the naming scheme, determining similarity can range from trivial to incredibly complex. For instance, if the naming scheme states that every newly discovered piece of malware should be assigned a consecutive number and have that number as a name, the process of determining similarity is reduced to determining the highest known malware number and generating the next one. Other naming schemes—e.g., based on date or place of discovery—require similarly simple classification algorithms. Unfortunately, as we shall explain in section 1.2, such naming schemes have other problems that make them unsuitable for a common malware naming standard.

The CARO Malware Naming Scheme is based on classifying malware into groups (called "families"), based on their code similarity. Unfortunately, determining code similarity is an extremely difficult task.

Again, partial solutions do exist for particular subclasses of malware. For instance, in the macro malware field we use a tool called `MIRA`, developed by Costin Raiu. This tool is run on a high–quality collection of macro malware, maintained by the author of this paper. The tool then builds a database of neural networks for recognizing the macro malware in the collection. This database, together with the tool, is made available to the anti–virus researchers. When a new piece of macro malware appears, the tool is run on it and it produces a list of the 10 most similar known pieces of macro malware, as recognized by the neural networks, together with some number reflecting the degree of similarity and ranging from 0 to 1. While this is not sufficient for a fully automatic macro malware classification process, it greatly reduces the amount of work for the anti–virus researchers. Once they get the output from the tool, they usually need to look manually at just a few (1–5) existing pieces of macro malware in order to determine whether the new one is "sufficiently similar" to any of them and whether it should be classified into an existing family.

A similar tool for script malware exists, although it is not as widely used, mostly because nobody is willing to spend the time and effort necessary to maintain a high–quality collection of script malware and routinely run the tool on it, in order to produce an up–to–date database of neural networks that can be used for automated classification of new script malware. (Contrary to some rumors, the author of this paper is only human and can do only so much work, not everything that nobody else feels like doing.☺)

The same principle is also applicable, to a certain degree, to non–parasitic binary viruses. Unfortunately, apart from some in–house experiments, no equivalent tool is widely available to anti–virus researchers at this time, not to mention the problems of maintaining a high–quality malware collection and regularly running the tool on it, in order to produce a database for automated classification.

An additional problem that makes automated classification of binary malware difficult is the fact that it often comes compressed with multiple executable compressors (e.g., UPX, Ice, Petite, etc.) for obfuscation purposes. One and the same stand–alone piece of malware (e.g., a particular Trojan) would look very different externally when compressed with different executable compressors (or with several of them). This makes automated classification of such malware even more difficult, because the various packing levels will have to be stripped first and even this is a very non–trivial probem.

1.1.6. Inability to Enforce a Particular Naming Scheme

No matter how good a naming standard, it is mostly worthless if nobody is using it. And, as experience has demonstrated, some anti–virus producers would follow their own malware naming scheme in royal disregard of any proposed standards.

In order to resolve this problem, some mechanism is needed for enforcing the chosen malware naming standard. Some kind of official body should be able to test how compliant any given product is to the standard and should have the power to penalize the producers of the non–compliant products and to force them to abide to the standard.

Unfortunately, at this time no such official body exists. CARO is certainly not "it". We are not even an official organization—we are just a group of friends with common interests and competence. We certainly do not have the power to enforce anything on anyone. In fact, many of us don't even have the power to impose a particular malware name on the companies we work for—let alone on anybody else.

Finding people willing to fulfil this "power vacuum" would not be difficult—the author of this paper is confident that various government bureaucrats are just itching for the job. Unfortunately, willingness alone is not sufficient, or the results would be disastrous. What is needed, above all else, is a very high level of competence and qualification in the field of anti–virus research. Even the ability to test the detection rate of known–virus scanners is clearly beyond the means of the average well–meaning person, as the hundreds of ridiculously incompetent and flawed published anti–virus product tests have unambiguously demonstrated. The level of competence required to determine whether a scanner not only detects a particular piece of malware but also names it correctly is even higher.

1.2. Alternate Malware Naming Schemes

In this subsection we shall examine the different computer malware naming schemes that have been tried in the past and found to be unsuccessful ([Bontchev98]).

1.2.1. Geographic Naming

In the early years of the virus era, when the viruses were few and far between, it seemed natural to name them after the place where they were first discovered—similarly to what is sometimes done for biological viruses. This is why, historically the very first naming schemes were geographical. Viruses like `Lehigh`, `Yale`, `Jerusalem`, `Vienna`, etc. were named after the places (universities or towns) where they were first discovered.

Unfortunately, such a naming scheme has serious drawbacks. In the modern world of electronic communications, malware programs often spread (or are distributed) faster than the information about their discovery. Because of this, it is common that one and the same malware program is given different names by the different anti–virus researchers who independently discover it in different geographical places. In fact, new malware often appears and spreads all over the world so fast (in a matter of hours) that it is impossible to determine even the *continent* of their origin—let alone the country or town.

Furthermore, the name itself contains no factual information that can be used to recognize the malware, so it is not very useful when two people have to understand that they are talking about the same malware. Finally, with the current boom in malware creation, it is very often the case that several different malware programs are discovered in one and the same geographical place, which again can lead to confusion.

1.2.2. Naming after the Infective Length

Another common–sense way of naming malware (and particularly viruses) is after their *infective length*, i.e., after the number of bytes they add to the infected objects. This naming scheme is much better than the geographical one. It is more objective, uses an important characteristic of the virus that can be used to identify it and so on. Unfortunately, it also has some drawbacks.

First of all, some viruses add a different number of bytes to the objects they infect, e.g., because they round the length of these objects up to the next multiple of 16, or add random garbage at the end, or for other reasons. *Second*, there are some kinds of viruses, e.g., the boot sector viruses, for which the infective length is not clearly and unambiguously defined. *Third*, many, often quite different, viruses have one and the same infective length. Finally, humans often have problems remembering information that is numeric only and are likely to make mistakes when reproducing it. This could lead to confusion when a user reports a virus infection.

1.2.3. Descriptive Naming

The next way to name malware is according to a description of its payload or of some other characteristic behavior. This is how names like "`Bouncing Ball`", "`Falling Letters`", etc. were constructed. Unfortunately, this naming scheme has its drawbacks too.

First, many viruses do not have a payload at all and do not do anything unusual. They just replicate in a rather straightforward way. *Second*, some quite different viruses share one and the same payload. For instance, the Yankee Doodle tune is played by both the `Old_Yankee` and the `Yankee_Doodle` viruses; the bouncing ball is displayed by both `Ping_Pong` and one of the `Murphy` viruses; the falling letters effect is used by both `Cascade` and the `Falling_Letters_Boot` viruses; turning the screen image upside–down is used by both `Flip` and `Mirror` viruses, and so on. *Third*, different anti–virus researchers could use different names to describe one and the same effect (e.g., Bouncing Ball vs. Ping Pong; Cascade vs. Falling Letters, etc.). *Fourth*, in many cases discovering the exact payload requires a careful analysis of the malware and this could delay its naming.

1.2.4. Naming after Some Text Found in the Virus

A simpler naming scheme is to name the malware after some text string found in its body. Just like the previous ones, this naming scheme does not suffer from a lack drawbacks.

First, many malware programs do not contain any characteristic text string. *Second*, many malware programs contain texts which are offensive, obscene, or libelous and are therefore inappropriate for common use as names. *Third*, it is often believed that using the same name as the author of the malware intended just boosts the malware creator's ego and should, therefore, be avoided.

1.2.5. Bezrukov's Naming Scheme

An interesting naming scheme has been developed by the Russian anti–virus researcher Nikolai Nikolaevich Bezrukov. According to it, each virus (it does not consider non–viral malware) is named by a one– to three–character identifier, indicating the types of objects that the virus infects, followed by the infective length of the virus, optionally followed by a single letter, indicating the particular variant, if more than one virus with the same identifier and infective length exists. For instance, a name like `RCE-1808A` means that this is a virus which is memory–resident (R); infects `COM` (C) and `EXE` (E) files; its infective length is 1808 bytes; and this is the first variant with such properties. Boot sector viruses are named in a similar way, with a single letter indicating whether they infect the MBR (P) or the DOS boot sector (B), followed by the contents (in hexa-

decimal) of the second byte of the infected boot sector—it is usually unique for most boot sector viruses.

Bezrukov's virus naming scheme is significantly more advanced than anything else described above. It allows the user to determine several important properties of the virus from its name alone. In addition to his naming scheme, Bezrukov also developed a scheme to compactly describe many other properties of the virus in a single and relatively short strings of letters and numbers. Such compressed descriptions can be further used by programs that automatically generate a natural–language textual description of the virus.

This naming scheme does not lack drawbacks either, but they are far less significant than the drawbacks of the previous naming schemes. Bezrukov's virus naming scheme has the following drawbacks. *First*, the virus names are relatively difficult for humans to remember. For instance, a name like `Cascade` is much easier to remember than the name `RC-1701A`. *Second*, with this naming scheme several completely different viruses can have a similar–looking name. For instance, one of the `Phoenix` viruses is also a memory–resident `COM` file infector with infective length 1701 bytes. Therefore, its name would be `RC-1701B` or something like this, regardless of the fact that the virus has absolutely nothing to do with the `Cascade` virus.

1.2.6. Numeric Naming

Another possible naming scheme is to use some sequence of numbers for identifying any particular piece of malware. The numbers can be derived from the date of discovery and/or could be simply incremented sequentially as new malware is discovered.

This naming scheme also has several major problems. *First*, such a number carries virtually no information about any properties of the malware (except perhaps about the date of discovery). To a certain degree, this problem can be alleviated by maintaining some kind of global reference database that describes the important properties of every known malware, given its numeric name. *Second*, two very similar malware variants could have very different names, which tends to be confusing. *Third*, most people have trouble remembering meaningless numbers, so they are likely to make mistakes when reporting which particular virus has infected their computer.

1.3. History of the CARO Malware Naming Scheme

The original CARO Naming Scheme was created at a meeting of CARO (the Computer Anti–virus Researchers Organization) in 1991, by a committee consisting of Friðrik Skúlason (FRISK Software International and then Virus Bulletin's technical editor), Dr. Alan Solomon (then from S&S International) and Vesselin Bontchev (then from the Virus Test Center, University of Hamburg). After several discussions, this committee decided that *the fundamental principle behind the naming scheme should be that malware (mainly viruses) should be grouped into families, according to the similarity of its programming code*. The committee also produced a document ([Bontchev91]), describing what a full malware name consisted of, according to this naming scheme, and how the various parts of it were to be constructed. While this was still humanly possible (i.e., when the number of known malware did not exceed 10,000), this committee met regularly at anti–virus conferences and also decided what the name of each known malware program should be, according to this naming scheme.

The first major revision of the Naming Scheme was completed in 2002 and a description of it was published in [FitzGerald02]. However, both Nick FitzGerald's paper and the original naming document were relatively difficult to find. This has drawn endless criticisms that a formal reference description of the Scheme was not publicly available and that this has prevented many anti–virus producers from following it.

Of course, such criticisms are ill–founded. Anybody who was really interested in a description of the Naming Scheme could obtain a copy of it, e.g., by asking one of its original authors. The *real* reasons for the malware naming mess are not the difficulty to find a description of the Naming Scheme but the ones described in section 1.1. Nevertheless, an easily accessible, up–to–date description of the Scheme should indeed be publicly available, and this is what the present paper aims to achieve. An HTML version of it will be made available, after the Virus Bulletin 2005 Conference, at

http://www.people.frisk–software.com/~bontchev/papers/naming.html

and will be constantly kept up to date, in order to reflect any future modifications. It already reflects the modifications that have been made in the Scheme since 2002.

2. Description of the CARO Malware Naming Scheme

The second part of this paper is dedicated to the description of the CARO Malware Naming Scheme.

2.1. General Format

As mentioned in section 1.3, the fundamental principle behind the CARO Malware Naming Scheme is that malware should be grouped into families, according to its code similarity. The other fundamental principle is that malware names should be *unique*—that is, every different malware variant, no matter how minor, should have a different name from that of any other malware.

The general format of a Full CARO Malware Name is

[<type>://][<platform>/]<family>[.<group>][.<length>].<variant>[<modifiers>][!<comment>]

where the items in square brackets are optional. According to this format, only the family name and the variant name of a piece of malware are mandatory and, as we shall see later, even the variant name can be omitted when reporting it. The Full Name is white space–delimited. That is, it cannot contain white space (i.e., space, tab, carriage return, line feed), and there is a white space before and after it.

In the following sections we shall describe the various parts of the full malware name in detail.

2.2. Malware Type

The *type* part of the full malware name in the CARO Malware Naming Scheme indicates, unsurprisingly, the type of malware is, e.g., virus, Trojan, etc. Currently, the Naming Scheme permits the following different types:

- `virus`. Basically, a virus is a program (or a set of programs) that can replicate itself recursively (i.e., the replicant is also a virus). For a formal definition see, for instance, [Bontchev98]. Note that whether the malware performs some other (e.g., destructive) action besides self–replication is

considered irrelevant for the purposes of determining its type. In some cases, the recursive replication cannot continue *ad infinitum* but stops after a certain number of generations. Such malware is also classified as a "virus", if the number of generations is larger than one; otherwise it is classified as an "intended" (see below). For macro viruses for platforms that use the concept of "global template" (e.g., Microsoft Word), a single "generation" is defined as infecting the global template from an infected document *and* then infecting a document from an infected global template, or, if the virus does not infect the global template, infecting a clean document from an infected one.

- `dropper`. This is malware that does not replicate itself but which releases self–replicating malware (i.e., viruses). It does not matter whether the virus is released on disk or only in memory, although in the latter case some anti–virus researchers prefer to use the term "injector". Normally, the family name of a dropper (see section 2.4) must be the same as the family name of the virus it releases. If, however, it can release more than one virus (the so called "multi–droppers"), it is acceptable to use a different family name or even to classify the malware differently (e.g., as a "tool" or as a "trojan"; see below).

- `intended`. An "intended" is malware written with the obvious intent to write a virus but which fails to replicate, usually due to some bug. Unfortunately, the definition of "intent" is highly subjective, so it is not possible to give a formal definition for this malware type.

- `trojan`. A "trojan" is malware that does not even try to replicate itself but which performs some intentionally destructive action, without correctly warning the user. Again, "intentionally", "destructive", "correctly" and "warns" are highly subjective terms. Consider, for instance, a disk formatting program that warns the user in Swahili that it is going to destroy the contents of the hard disk and which assumes that the default answer is "yes". Is such a program a Trojan or not? So, no formal definition of this malware type is possible.

- `pws`. A "password stealer" is a program, the main purpose of which is to steal passwords. Often (but not always) this is achieved via some kind of keyboard logging. Some anti–virus researchers prefer to classify such programs as "trojans", but CARO has decided that a special malware type for them is needed in the Naming Scheme.

- `dialer`. This is a program that installs itself in the chain of programs invoked when the computer is establishing a dial–up connection. The purpose of such a program is to force the connection to the Internet to go through a particular premium phone number. Not all programs of this type are malicious, some are used quite legitimately for micro–payments. The vast majority of them *are* malicious, though; their only purpose is to steal money from the victim by forcing them to dial a particular premium phone number. Whether a dialer is non–malicious is determined by whether it properly informs the user of its actions and whether it is easily uninstalled. Again, some anti–virus researchers prefer to classify the malicious "dial-

ers" as "trojans", but CARO has decided that a special malware type for them is needed in the Naming Scheme.

- `backdoor`. This is a program that allows access to the machine on which it has been installed, access that circumvents the legitimate login authentication procedures for that machine. Note that it is perfectly possible for a "backdoor" to use a login authentication procedure on its own so that only a particular attacker is granted access to the compromised machine; not just anyone. Not all backdoors are installed by external attackers, sometimes a system program shipped with the machine can be a backdoor, e.g., because the vendor has forgotten to disable some undocumented way of accessing the machine, one that had been put there originally for debugging purposes. Again, some anti–virus researchers prefer to classify such programs as "trojans", but CARO has decided that a special malware type for them is needed in the Naming Scheme.

- `exploit`. An "exploit" is a way of bypassing the security of a program or an operating system, usually because of some kind of bug. Programs that demonstrate such security flaws are also called "exploits". It is highly recommended that Mitre's CVE/CAN vulnerability names ([Mitre]) are used as "family names" (see section 2.4) when reporting exploits.

- `tool`. A "tool" is a program that is not dangerous to the user who runs it, but that can be used to produce malicious programs or to perform malicious actions. A typical example is a virus construction kit, a program for automated construction of new computer viruses. (It does not matter whether they are constructed in ready–to–execute form or only in source.) In the past, the Naming Scheme used to have the malware type "kit" which meant exactly that. However, it was decided to rename this malware type and to extend its meaning to cover construction kits for other kinds of malware, password cracking tools, and all other kinds of tools used by the attackers.

- `garbage`. The type "garbage" is reserved for the various programs that do not perform any meaningful action (usually due to bugs) and do not even try to be viruses (or they would be classified as "intended") but which tend to float around in the various low–quality virus exchange collections and which are often included in the test sets used by incompetent testers to test virus scanners. As a result, many vendors have given up and decided that it is more cost–effective to implement detection of them instead of educating the testers.☺ In some special cases buggy viruses can produce non–replicable replicants; these should also be classified as "garbage".

In some cases, a piece of malware matches the definitions of several of the permitted types described above, e.g., it can be both a "virus" (in the sense that it replicates itself) and a "dropper" (in the sense that it drops another virus). In such cases it should be classified as the worst type, the definition of which it matches. The permitted malware types are listed above in such an order, with "virus" being the worst. Therefore, in our particular example, the "virus and dropper" malware should be classified simply as "virus".

Currently, the above malware types are the *only* malware types permitted by the CARO Malware Naming Scheme. Notably, there is no special malware type for "worm; these should be classified as "viruses". The reason for this is that it seems impossible to reach an agreement among anti–virus researchers on what, exactly, a worm is. There are at least three fundamentally different definitions of this term and different anti–virus researchers prefer different definitions. In order to avoid confusion, the Naming Scheme does not use such a malware type at all. If an anti–virus producer feels that they absolutely must report that something is a worm (according to their pet definition of this term), they should put this information in the comment field (see section 2.9).

In addition, there are no malware types for "spam", "adware", "spyware", "phishing scam", "non–malicious application" or "unwanted application", despite the fact that some anti–virus vendors have chosen to report such things with their products. Although CARO has considered proposals for adding special malware types for these to the Naming Scheme, it was decided that either the definitions of these terms were too imprecise or there was insufficient need for them (e.g., because it is not really the job of an anti–virus program to report such things). However, malware types might be introduced for them in the future (and/or for other things as well). When/if this happens, it will be reflected in the on–line version of this document.

2.3. Platform

The *platform* part of the full malware name in the CARO Malware Naming Scheme specifies the platform on which the malware works. This can be an operating system (e.g., "PalmOS"), a set of operating systems (e.g., "Win32"), an application (e.g., "ExcelMacro"), or a language interpreter (e.g., "VBS"). It is *not*, however, a file type. For instance, the platform of a virus written in the Visual Basic Script language must be "VBS", even if that particular virus is designed to reside only in HTML or only in CHM or only in PDF files.

Each platform name exists in two forms, a long form and a short form, although in some cases the two forms can be identical. Either of the two forms can be used when reporting some malware, although in practice most producers tend to prefer the short form.

A complete list of the platforms allowed by the CARO Malware Naming Scheme is given in Appendix A. Platforms exist, which are not listed there, for which malware is possible, but for which no known malware yet exists. These platforms are intentionally not listed, in order not to encourage the virus writers. CARO has decided on names for some of them, but these names will not be made public until malware for the respective platform appears.

The platform "`DOS`" is the default platform and should be omitted when reporting malware.

Some malware can work on more than one platform. For instance, there are macro viruses that infect both Microsoft Word (the "`W97M`" platform) and Microsoft Excel (the "`X97M`" platform). In other cases malware can consist of multiple components, each component written for a different platform. For instance, a script virus could consist of both VBScript and JavaScript components, all of them co–existing in the same infected HTML document. Or a Win32 worm could drop script and/or macro components, which might not even be viral by themselves, but which ensure

the replication (or at least the activation) of the Win32 executable. There are several ways to handle such cases.

The most correct, formal way is to list alphabetically all the relevant platforms between curly braces and separated by commas, e.g.,

```
virus://{VBS,W97M,Win32}/Foo.A@mm
```

However, this makes the full name look rather clumsy and many products have problems handling such long malware names. Therefore, the Naming Scheme permits the use of the artificial platform "Multi" in such cases. For multi–platform macro viruses only, the artificial platform "O97M" is also a permitted abbreviation, although it is deprecated in favor of "Multi".

Finally, reporting multi–platform malware with just the platform on which it has been detected, is permitted. For instance, the VBScript component of the imaginary virus mentioned above may be reported as "`VBS/Foo.A`", the Word97Macro component may be reported as "`W97M/Foo.A`" and the Win32 component may be reported as "`W32/Foo.A`". Similarly, a macro virus that infects both Word documents and Excel spreadsheets may be reported as "`W97M/Bar.A`" when found in Word documents and as "`X97M/Bar.A`" when found in Excel spreadsheets, even if it looks exactly the same in both places.

2.4. Family

The *family name* is the only part of the full malware name that a virus scanner must report when it believes that it has detected the malware. One of the fundamental principles of the CARO Malware Naming Scheme is that malware should be grouped into families, according to the similarity of its code. This is useful to the developers of anti–virus software, because malware that it programmed in a similar way usually needs similar methods of detection and removal. So, the fact that, e.g., a new virus is classified into a particular known family conveys to the anti–virus researchers the useful hint that some of the detection and disinfection methods for the other, already known members of that family might be applicable (possibly with some modifications) to the new virus too.

2.4.1. General Format

The family name is constructed from the characters of the character set `[A-Za-z0-9_-]` that is, upper– and lowercase letters, digits, underscore and minus. No other characters are allowed in a family name. In particular, the characters '`%`' and '`&`' are not allowed. If somebody wants to include the words "percent" or "and" in a family name, they should use "`_Pct_`" (or "`_Pct`", if at the end of the family name) and "`_And_`".

Spaces are also not allowed in a family name (or, in fact, anywhere in a full malware name). Those who want to use multi–word family names should use underscore characters instead of spaces. Alternatively, they can simply join the words together and use uppercase letters for the first character of every word and lowercase letters for the remaining characters of the words. For instance, "`My Party`" is not a valid family name (because it contains a space) but both "`My_Party`" and "`MyParty`" are valid family names. Using more than one underscore characters consecutively (e.g., "`My__Party`" is strongly discouraged for aesthetic reasons.

Furthermore, the letter case is considered irrelevant when comparing two different family names—so, "`MyParty`", "`myparty`", "`MYPARTY`" and "`mYpArTy`" are one and the same family name although the first form is "best" for aesthetic reasons. This rule was introduced in the naming scheme in order to accommodate some anti–virus products that had the limitation that they could only report the names of detected malware in uppercase.

One final limitation is that the family name must not be more than 20 characters long. In fact, even this limit is considered rather large. The only reason why it was introduced was for historical reasons, in order to accommodate monstrously long family names like "`Green_Caterpillar`". In practice, shorter names are preferred, although this suggestion should not be taken to the extreme; i.e., do not shorten the family name to the point of unintelligibility.

2.4.2. Rules for Constructing Proper Family Names

While the construction of the other parts of the full malware name in the CARO Malware Naming Scheme is more or less obvious, the construction of the family name (even while adhering to the formatting rules listed in the previous section) is given to "artistic interpretation". It is not possible to provide formal rules for constructing the family name of a new piece of malware that is not related to any of the already existing malware families. All we can do is provide some guidelines on how to construct good family names.

Don'ts…

- Do not use company names, brand names, or names of living people. Using the name of the malware author is permissible but only when the malware is provably written by that person, and even then it is discouraged. Common first names are also permissible, but should be avoided, if possible. It is explicitly forbidden to use names associated with the anti–virus and computer security industry.
- Do not use an existing family name unless the malware indeed belongs to that family by code similarity. Note that this applies solely to the family name part of the full malware name. Using the same family name for a VBS virus as an existing Word97Macro virus or Win32 Trojan is wrong unless there is good reason to consider them closely related (there are strong code similarities, they are different components of the same malware for different platforms, etc). Being on different platforms alone does not justify the re–use of a family name.
- Do not invent a new family name if there is an existing, acceptable name. This is the inverse of the previous rule. If a new piece of malware is 'clearly' a new member of an existing family, put it in that family. Just because it has a different payload or trigger condition or is even attributed to a different writer, this does not necessarily justify the creation of a new family.
- Do not use obscene or offensive names. Following this rule for the English language is a must. It is recommended that it is followed for other languages as well, but, naturally, the anti–virus researcher naming a new malware family cannot be expected to know whether a text string found in the malware is some obscenity in some obscure language. Also, "offen-

sive" is very difficult to define since different things are offensive to the different people and cultures. In any case, in order to err on the safe side, names with religious meaning should be avoided, since this is a touchy subject in many cultures.

- Do not assume that just because a sample arrives with a particular name, that this is its correct name. This is especially important when processing presumably sorted collections from other researchers.
- Do not use numeric family names. There is one exception to this rule; see section 2.4.3.
- Do not use as family names words that are used in the other parts of the full name. For instance, "`W97M`" is an invalid family name, because it is a standard platform name.
- Do not use words that are too common, too generic and/or non–descriptive. For instance, "virus", "network", "infector", etc. are *extremely* bad family names.

Dos…

- Avoid the name suggested by the author of the malware writer. When no obviously better name is possible, it is acceptable to "pervert" (e.g., misspell or reverse) the name suggested by the author.
- Avoid naming malware after a file that traditionally or conventionally contains the malware. Many mass–mailing viruses send themselves with a fixed filename for the attachment, but a later variant of the same family might use a different filename and become much more widespread and better known.
- Avoid naming a family after the activation date of the payload of some variant (e.g., "`Friday_the_13th`" is a bad family name). The reason for this rule is that the activation date is likely to be different (it's one of the easiest things to change) in other variants of the same family and the name would be misleading for them.
- Avoid geographic names which are based on the discovery site. The same malware might appear simultaneously in several different geographic places.
- If multiple acceptable names exist, select the original one, the one used by the majority of existing anti–virus programs or the more descriptive one. Generally, the name chosen by the researcher who first 'isolates' a piece of malware has primacy, unless it is shown to be a poor choice. In such cases agreement on a better name should be sought by discussion between researchers.

2.4.3. Special Family Names

With 200,000 different known malware programs belonging to hundreds of different families, it is not always easy to pick a good new family name. It is especially annoying to do this for every new, trivial, do–nothing–in–particular virus that appears and happens to be sufficiently dissimilar to all the known virus families. In order to alleviate this problem a little, several artificial virus families have been cre-

ated. The members of these families are not grouped by code similarity. Instead, they are grouped together by a few particular important properties.

It should be noted, however, that these artificial families apply mostly to 16–bit DOS viruses. As such, they are mainly of historical interest. Nowadays, the vast majority of new malware programs are 32–bit Windows programs and, of them, about 95% are non–viral malware (mostly various Trojan horses, backdoors, keyloggers, password stealers, etc.). Therefore, the above artificial families are very rarely used for contemporary new malware.

The artificial virus families and the common properties of their members are listed in the following table:

HLLC	Companion viruses written in some high–level language
HLLO	Overwriting viruses written in some high–level language
HLLP	Parasitic viruses written in some high–level language
SillyB	Trivial (small, with no payload) viruses that infect the DOS Boot Sector of the hard disk
SillyC	Short, non–memory–resident viruses with no payload that infect parasitically and only COM files
SillyCE	Short, non–memory–resident viruses with no payload that infect parasitically both COM and EXE files
SillyCER	Short, memory–resident viruses with no payload that infect parasitically both COM and EXE files
SillyCR	Short, memory–resident viruses with no payload that infect parasitically and only COM files
SillyE	Short, non–memory–resident viruses with no payload that infect parasitically and only EXE files
SillyER	Short, memory–resident viruses with no payload that infect parasitically and only EXE files
SillyOR	Short, memory–resident overwriting viruses with no payload
SillyP	Trivial (small, with no payload) viruses that infect the Master Boot Record of the hard disk
Trivial	Short (100 bytes of code or less, text messages excluded) overwriting viruses with no payload

Another special case is parasitic viruses that have not yet been analyzed and for which no obviously good family name has been selected. Such viruses may be assigned “temporary” family names consisting of their infective length, preceded by an underscore, e.g., _1234. However, every effort should be made to avoid such temporary names. If they have to be used, every effort should be made to analyze the virus as soon as possible, select a proper family name for it, and rename the temporary family name to the newly selected one.

2.4.4. Malware Relationship

As already mentioned several times, the CARO Malware Naming Scheme is based on grouping malware into families, according to its code similarity. Unfortunately, as mentioned in section 1.1.5, there are no reliable, automated means for determining, in the general case, whether newly discovered malware belongs to a known malware family and which one. In this section we shall try to give at least some guidelines to follow when determining malware relationship ([Bontchev91]).

The main idea behind this method of malware classification is to make sure that

a) Non–related pieces of malware belong to separate malware families and

b) Related, but different malware that can be disinfected in exactly the same way are classified as different variants of one and the same malware family.

In order to achieve the above, the following rules are applied when considering the relationship between any two pieces of malware, A and B:

Rule #1: If the malware is encrypted or packed in any other way, decrypt and/or unpack it and consider the decrypted body when applying the following rules. If the malware is polymorphic, do not consider the variable part (i.e., the decryptor) in the comparison. Also, during the comparison, ignore the contents of any data areas present in the malware. Unfortunately, the latter usually requires manual analysis of the malware and cannot be done fully automatically, although some aspects of it can be automated.

Rule #2: If there are fundamental structural differences in how A and B infect the same program (or replicate; if they are not parasitic viruses)—for example, if one of the two viruses appends its code at the end of the infected files, while the other one prepends its code at the front of them—then the viruses belong to two different families. However, this rule *does* permit A and B to be classified in the same family, even if one of them infects only `COM` files, while the other one infects both `COM` and `EXE` files, provided that they infect these files in a structurally similar way.

Rule #3: If the function

Related (A, B)

returns TRUE, then the viruses A and B belong to the same malware family. The function *Related (x, y)* is described below.

Rule #4: If there exists a malware X such that the expression

Related (A, X) AND Related (B, X)

is true, then the two pieces of malware, A and B, belong to the same virus family as X.

Rule #5: If there exist two pieces of malware, A' and B', where the following holds true

(A' and B' are known to belong to the same family) AND
Related (A, A') AND Related (B, B')

then the two pieces of malware A and B belong to one and the same malware family (the same malware family to which A' and B' belong). Otherwise, A and B belong to two different malware families.

Rule #6: If the two pieces of malware, A and B, belong to one and the same malware family but have different infective length or cannot be disinfected in exactly the same way, then they belong to two different variants of this malware family.

The function *Related (X, Y)* compares two blocks of code X and Y and returns TRUE if the block X has a significant amount of code in common with block Y. The function is implemented in the following way:

```
Related (X, Y) ::= Average (Substrings (X, Y, N) / (Length (Y)—N + 1),
                            Substrings (Y, X, N) / (Length (X)—N + 1)) > LIMIT;
```

where

Substrings (u, v, t) is the number of all substrings of *u* of length *t* found within *v*. For N about 12–16, the value of LIMIT is usually 0.5–0.6, this is enough to detect the viruses which are related. For unrelated viruses, the expression on the left side of the greater–than sign usually evaluates to 0.05 or less.

2.5. Group

The *group* part of the full malware name in the CARO Malware Naming Scheme is used when a large subset of a malware family contains members that are sufficiently similar to each other and sufficiently different from the other members of the same family, yet at the same time the members of this subset are not similar enough to each other to be classified as variants. A typical example are the `AutoCAD` viruses which are clearly a clone of the `Jerusalem` virus.

The rules for constructing a group name are the same as those for constructing a family name.

Group names are preserved mostly for historical purposes, because removing them would involve renaming a lot of well–established malware names. However, their use is deprecated and it is strongly recommended that they are not used when new malware (that doesn't belong to the already existing groups) is discovered.

2.6. Length

The *length* part of the full malware name in the CARO Malware Naming Scheme indicates the infective length of the particular piece of malware. It makes sense to use it only for viruses that infect parasitically, although, for historical reasons, it is also used for some small companion viruses.

Even in the case of parasitic viruses, their infective length is not always easy to determine. For instance, the original `Jerusalem` virus prepends 1808 bytes at the beginning of the COM files it infects, but it appends 1813 bytes at the end of the EXE files it infects. Other viruses pad with random bytes the size of the files they infect to some multiple (usually 16), so they do not always increase the size of the infected files by one and the same number. Some kinds of parasitic viruses, the so called "cavity" viruses, overwrite parts of the files they infect that originally contained zeroes (or some other constant). Although such viruses have an infective length, this is not immediately obvious to the user, since the infected object does not increase in size by that length.

Originally, the infective length was made part of the full malware name because it reported an important property of the malware, the size by which the infected files increased. This made it both easier for the user to recognize the infection (at that time many people used to remember the original sizes of important system files such as `COMMAND.COM`) and served as an important prompt to the developer of anti–virus software, because it indicated how many bytes had to be removed from the infected file during disinfection.

Nowadays this is no longer a significant factor. There are thousands of system files on the users' machines and hardly any user remembers their exact size. The most successful viruses nowadays are self–contained worms that do not infect parasitically. The size of the viruses used to be a few hundred bytes to a few kilobytes—i.e., a number easily remembered—while nowadays viruses that are dozens to hundreds of

kilobytes are the rule, not an exception. In addition, they are often compressed with some kind of executable compressor (e.g., UPX, Ice, etc.) and their exact size is meaningless both to the user and to the developers of anti–virus software, who are now more concerned with implementing detection of the malware instead of surgical disinfection, reasoning that detection alone is sufficient for the on–access scanner (or the e–mail scanner) to protect the machine from becoming infected in the first place, so the ability to disinfect is of secondary importance. Finally, these days, viruses (let alone parasitic viruses) are not even the most widespread kind of malware, various kinds of Trojan horses appear much more often. This is why using the infective length in newly discovered malware should be avoided, unless there is a good reason not to.

In particular, the infective length is not used in macro and script viruses, in viruses written in a high–level language, in non–parasitic viruses, in Win32 viruses, in viruses that have a very large infective length (more than 50 Kb) and in non-viral malware.

There is one exception in the case of macro viruses. When the virus is a multi–component virus and one of its components has an infective length, that length is used when naming the macro component too, even if it doesn't apply directly to it. For instance the infective length is used in the name of the `{W32,W97M}/Beast.41472.A` virus, even though the size of its `Word97Macro` component is not 41472 bytes long.

The length, when used, is always a number, i.e., it can contain only decimal digits.

2.7. Variant

The *variant* part of the full malware name in the CARO Malware Naming Scheme is used to distinguish between different malware programs that belong to the same family (and to the same group, and have the same infective length, when these parts of the full name are present).

For malware that does include an infective length component in its full name (i.e., for relatively small parasitic binary viruses), the variant names usually serve to distinguish between minor patches. For all other kinds of malware (non–viral malware, non–parasitic viruses, macro and script viruses, etc.) the variant names are the main hierarchical level at which different members of the same family are distinguished.

2.7.1. Variant Naming

The variant name consists of upper case letters, assigned consecutively as each new variant in a malware family is discovered. "Consecutively" means that the first variant of a malware family always has the variant name `A`, the next one—`B` and so on. When the variant name `Z` is reached, the next discovered malware program in that family is assigned the variant name `AA`, then `AB`, etc. till `AZ`, then `BA`, `BB`, … `BZ`, `CA`, `CB`, … `CZ`, … `ZZ`, `AAA`, `AAB`, … `ZZZ`, `AAAA`, and so on.

Note that the order in which the variant names are assigned reflects the order of their discovery, *not* the apparent order of their creation or any other order. This is the main point where naming confusion occurs between the different anti–virus products. While new families are discovered relatively rarely and there is usually time to synchronize their names among the anti–virus companies that are willing to make the

necessary effort, new variants appear very frequently (sometimes several per day) and, given the lack of precise identification tools mentioned in section 1.1.4, it is often too problematic to achieve variant name synchronization among the different anti–virus products. Any serious attempts to reduce the malware naming mess should concentrate on this issue.

2.7.2. Variant Reporting

Another important point to note is that the variant name (together with the family name) is a mandatory part of every malware name in the CARO Malware Naming Scheme. Every other part (type, platform, length, modifiers, comment) can be missing, but the family name and the variant name *must* be present, even if there is only one known variant that belongs to the respective family. That is, if the family `Foo` consists of only a single member, its name should be `Foo.A` (or `Foo.1234.A`, if the infective length part makes sense for it and if it is 1234 bytes) and not just `Foo` (or `Foo.1234`).

However, when anti–virus products report malware, they are allowed (and even encouraged) to omit the variant name part, if they are unable to distinguish between the different variants in the same family. That is, if a scanner is unable to distinguish between the `Foo.A` and `Foo.B` viruses, it should report just `Foo`. The same rule also applies to the group name and the infective length, if they are part of the full name of the malware. That is, if a scanner is unable to distinguish between different group members in a family, it should report only the family. Similarly, if it is unable to distinguish between viruses with different infective lengths, it should report only the family part of their names (and also the group part, if it is present and if it can distinguish between viruses that belong to different groups of the same family).

There is one exception to the above rule, however. If a scanner is unable to distinguish between a particular subset of variants but can identify precisely enough the other variants in the family and is sufficiently confident that the malware it has found can be only one of the subset it cannot identify precisely, the scanner is allowed to report the whole subset of variants.

To make the above clear, let us consider a particular example. Let us suppose that a virus family, `Foo`, contains the variants `A`, `B`, `C`, `D` and `E`. Let us also suppose that a particular scanner is unable to distinguish between the variants `A`, `B`, `C` and `E`—but can identify the variant `D` exactly. It should, obviously, report the `D` variant as `Foo.D` because it can identify it exactly. When reporting any of the other variants, though, the scanner is allowed the following alternatives. *First*, it may report only the family name—`Foo`. *Second*, it may report each and every one of the variants `A`, `B`, `C` and `E` either as `Foo.{A,B,C,E}`, or as `Foo.{A-C,E}`.

2.7.3. Devolutions

There is one exception to the rule that variant names consist of uppercase letters only. Currently it applies only to macro viruses but there are no fundamental reasons why the need for it could not arise for other kinds of malware as well.

In the general case, a macro virus consists of a set of macros (although in many cases the set contains only one element). Many macro viruses can replicate in more than one way, e.g., on document open, on document close, on document save, etc. Some macro viruses contain bugs and simply forget to copy some of their macros when replicating in a particular way. This is called *devolution* (the opposite of evolu-

tion). However, the resulting subset of macros can also be viral. But, since it would be a set of macros different from the original set, it would be a different virus. It would still belong to the same virus family, obviously, but it would have to be assigned a different variant name.

In such cases, the CARO Malware Naming Scheme states that instead of assigning the next consecutive variant name, the new variant name should be formed by taking the original variant name and appending a number to it. For instance, if the virus `W97M/Foo.A` devolves under some circumstances, then the devolved variant must be named `W97M/Foo.A1`, and *not* `W97M/Foo.B`. It is possible that the original variant devolves in more than one way, or that the first devolution devolves further. In such cases, consecutive numbers should be used: `W97M/Foo.A2`, `W97M/Foo.A3`, etc. It is also possible that two different variants produce one and the same devolution, i.e., that `W97M/Foo.A` and `W97M/Foo.B` both devolve to `W97M/Foo.A1`. A typical example of devolving viruses are the viruses from the `WM/Rapi` virus family ([Bontchev97]).

The selection of numbers should be consecutive, should start from 1, and should reflect the order in which the devolutions are discovered. It does *not have to* reflect the hierarchy of devolution, i.e., the numbers carry no information whether the variant has devolved from the original variant (and from which one, if two different variants produce the same devolution) or from some intermediate devolution. Note also that the original variant is *never* assigned a number, its variant name consists of letter(s) only. The devolution number must *never* be used by itself, it *must* be used *only* as part of the variant name. For instance, the name `WM/Foo.1` is incorrect specification of a devolution; it should be, e.g., `WM/Foo.A1`.

A scanner should report the devolution number *only* if it is capable of distinguishing between the main variant and its devolutions.

Finally, the following rules are used to determine whether a subset of the macros of the original variant are a devolution or not. *First*, the subset must be produced in a "natural" way, i.e., during the natural replication of the virus, and not by artificially removing some of its macros, either manually or during (improper) disinfection. If the removal of macros is not produced naturally, the resulting subset is considered a new variant, not a devolution. A typical example is the `WM/Dzt` virus family, where the `B` variant is produced from the `A` variant by some scanner incorrectly removing only one of the `A` variant's macros ([Bontchev97]).

Second, the subset must be incapable of reproducing the original set. If a virus can, during natural replication, reduce the set of macros it consists of, but the reduced set can later (also during natural replication) return to the original set, then the reduced set is still considered to be the original variant, it *must not* be given a new variant name or a new devolution number. A typical example of such viruses are the viruses in the `WM/Johnny` family.

2.8. Modifiers

The *modifier* part of the full malware name in the CARO Malware Naming Scheme lists some properties of the malware, that are deemed important enough to be conveyed to the user immediately (i.e., at the time when the malware is reported).

2.8.1. General Format

The general format of the modifier part is:

[:<locale>][{@<at_modifier>}]

An earlier version of the CARO Malware Naming Scheme also allowed modifiers for specifying the packer(s) with which the malware was compressed or the polymorphic engine used by it. However, those modifiers were not used by anyone and it was decided to remove them from the Naming Scheme.

2.8.2. Locale

The *locale* part is currently used only for macro malware, although there are no reasons why, theoretically, other kinds of malware might appear that requires it. It is used to indicate the language version of the platform, which language version is *required*, in order for the malware to execute properly.

There are a few things that should be emphasized about the locale. *First*, it is used to indicate a *required* language version of Microsoft Office, not a *supported* one. For instance, if some macro malware has special code that allows it to run under the German version of Microsoft Word, it will *not* get the `:De` locale identifier if it is also runs under the English version of Word.

Second, the English language version is the default and is not denoted with any special locale identifier.

Third, the locale identifier specifies a *language version* of an Office product, *not* a country and not even a language. For instance, the `:Tw` locale identifier is used because there was a Taiwanese version of Office, not a Chinese one.

Fourth, no distinction is made between the various minor language variants (e.g., French vs. Canadian French, or Portuguese vs. Brazilian Portuguese, or Simplified Chinese vs. Mandarin).

Currently, the following locale identifiers are used:

Identifier	Language
Br	Brazilian
De	German
Es	Spanish
Fr	French
He	Hebrew
It	Italian
Jp	Japanese
NL	Dutch
PL	Polish
Ru	Russian
Th	Thai
Tw	Chinese

If the need for more locale identifiers arises (e.g., because malware appears that runs properly only on some other language version of some platform), the above table will be extended.

If some malware can run on more than one of the language versions listed in the above table (but cannot run under the English language version), it is acceptable to use more than one locale identifier, separated by commas and surrounded by curly braces like this: `virus://WM/Foo.A:{De,Fr}`. In such cases the locale identifiers should be listed in alphabetical order. Currently no malware exists that would require this notation to be used, though.

In some cases (e.g., script malware) the malware might work only if existing in Unicode (not regular ASCII) form. In such cases it is recommended to use the special locale identifier `:Uni`. No such malware currently exists, though. Although several existing script viruses spread in Unicode form, there is nothing in their code that would prevent from working in ASCII form too.

2.8.3. At Modifiers

The *at modifier* part lists some properties of the malware which are deemed critically important to report to the user as soon as the malware is discovered. This is usually because the malware has some fast–spreading properties that would require a higher priority of dealing with it than with any other kind of malware.

Originally, only the `@mm` modifier was allowed (see below for its description). Currently several new ones have been added to the list. It is important to understand that the list is dynamic. In the future CARO might decide to add additional modifiers or to remove some of the existing ones.

It is possible that one and the same piece of malware has properties covered by more than one modifier. In such cases the properties should be listed in alphabetical order. Unlike the other cases when multiple kinds of the same part of the malware name are used, in this case the different modifiers are not separated by commas and are not enclosed in curly braces. Example: `virus://W97M/Foo.A@irc@mm`.

The modifiers currently allowed are described in the table below.

Modifier	Description
@exp	Malware that relies on the existence of some exploit or vulnerability, in order to work properly.
@i	An Internet worm, a virus that spreads directly from one computer to another over the Internet not by using e–mail but by some other means. The `CodeRed` virus is a typical example.
@irc	A virus that uses IRC to spread. Note that this does not necessarily mean that the virus is written as an IRC script. Use the IRC or PIRC platform name to indicate the latter.
@m	A slow–spreading mass–mailer. Such viruses usually e–mail themselves one at a time, e.g., as a response to a received e–mail message.
@mm	An explosively spreading mass–mailer. Its execution usually results in the virus mass–mailing itself to all e–mail addresses it can find.
@p2p	A virus designed to spread over the peer–to–peer networks.
@s	A virus designed to spread via open network shares, i.e., shared network drives that do not require a password to access them. Note that this is very different from using a peer–to–peer network, although some people tend to equate P2P networks with “sharing”.

It is important to note that a modifier should be used *only* if the malware does indeed have the corresponding property, not when it just contains code for it. For in-

stance, a virus that contains mass–mailing code but which does not mass–mail itself (e.g., because the mass–mailing code has a bug or because it is never invoked) *must not* be given the `@mm` modifier.

2.9. Comment

Although great pains have been taken by the developers of the CARO Malware Naming Scheme in order to ensure that it is acceptable to all anti–virus producers, there will always be things that a particular producer would like to report about a particular malware that are not included in the Scheme. For instance, some producers want to specify that a particular virus is a worm, others want to specify that a virus is corrupted, or compressed with something, or has a particular property, and so on.

In order to accommodate such needs, the Scheme allows any kind of comment to be included after the name, separated from it with an exclamation mark. Technically, the comment is not part of the full malware name; it is only a reporting feature. It can contain almost anything—the only restrictions are that it does not contain white space and that it is present (after an exclamation mark) as the rightmost part of the name. Every anti–virus producer is free to include in the comment whatever information they want, in whatever format they want (provided that it does not include a white space). In particular, it *may* contain characters that are otherwise not allowed in the other components of the full malware name or that are used as separators, including commas, dots, exclamation marks, etc.

3. Conclusion

In this paper we have explained why the malware naming confusion exists. We have also described shortly the various alternative, less successful malware naming schemes and have given a full and detailed description of the current status of the CARO Malware Naming Scheme. It is our hope and belief that the wide availability of this paper will help reduce the existing malware naming confusion.

4. References

[Bontchev91] Vesselin Bontchev, Friðrik Skúlason, Dr. Alan Solomon, "Virus Naming Scheme", available electronically from ftp://ftp.informatik.uni–hamburg.de/pub/virus/texts/tests/vtc/pc–av/1994–07/naming.zip.

[Bontchev98] Vesselin Bontchev, "Methodology of Computer Anti–Virus Research", Ph.D. thesis, University of Hamburg, 1998.

[Bontchev97] Vesselin Bontchev, "Macro Virus Identification Problems", Proc. 7th Int. Virus Bull. Conf., 1997, pp. 175–196.

[Bontchev04] Vesselin Bontchev, "Anti–Virus Spamming and the Virus Naming Mess—Part 2", Virus Bull., July, 2004, pp. 13–15.

[FitzGerald02] Nick FitzGerald, "A Virus by Any Other Name: Towards the Revised CARO Naming Convention", Proc. AVAR'2002 Conf., Seoul, 2002, pp. 141–166.

[Mitre] http://cve.mitre.org.

Appendix A—List of Permitted Platform Names

The following table lists all currently valid platform names, according to the CARO Malware Naming Scheme. Malware for other platforms is possible but doesn't exist yet. CARO has agreed on several platform names for such platforms but they are not made public, in order not to encourage the creation of malware for these platforms. If/when malware for them appears, their official names will be made publicly available.

If you have discovered malware and are not sure what its platform is, or think that it is for a platform not listed in the table below, please contact a CARO member and discuss the platform name with him/her instead of trying to be creative and inventing your own platform name.

Short Form	Long Form	Comments
A2M	Access2Macro	Macro malware for Microsoft Access 2.0
A97M	Access97Macro	Macro malware for Visual Basic for Applications (VBA) for Access, that shipped in Access 97 and later.
ABAP	ABAP	Malware for the SAP /R3 Advanced Business Application Programming environment.
ACM	AutoCADMacro	VBA macro malware for AutoCAD r14 and later.
ActnS	ActionScript	Requires the Macromedia ActionScript interpreter found in some ShockWave Flash (and possibly other) animation players.
AM	AccessMacro	Macro malware for AccessBasic—an alternative macro language for Microsoft Access.
AmigaOS	AmigaOS	Malware for the Amiga computers.
AplS	AppleScript	Malware for the AppleScript interpreter on Macintosh computers.
APM	AmiProMacro	Macro malware for the AmiPro editor.
Apple2	AppleII	Malware for the Apple][,][+, //, //e and //c computers.
AutoLISP	AutoLISPScript	Malware written in the LISP dialect used in AutoCAD.
BAT	BAT	Malware that requires a DOS, Windows or NT command interpreter or close clone (e.g., 4DOS or 4NT).
BeOS	BeOS	Malware for BeOS.
Boot	Boot	Malware that resides in the Master Boot Record or the DOS Boot Sector of the computer.
BSD	BSD	Malware specific to BSD–derived platforms. *Unix* is still the preferred platform name.

C9M	Corel9Macro	VBA macro malware for Corel Draw! Version 9.0 and later.
CSC	CorelScript	Malware for the CorelScript interpreter in many Corel products.
DCL	DCLScript	Malware written in the DCL scripting language.
DOS	DOS	Infects DOS COM and/or EXE (MZ) and/or SYS format files and requires some version (*any* version) of MS–DOS or a closely compatible OS (PC–DOS, DR–DOS).
DSOS	DSOS	Malware for the Nintendo DS platform.
EPOC	EPOC	Malware for the EPOC OS before version 6.
HLP	WinHelpScript	Malware for the script interpreter of the WinHelp display engine. (Note: this is *not* the correct platform for *JS* or *VBS* script malware embedded in HTML and 'compiled' into CHM help files.)
IDAS	IDAScript	Malware written in the scripting language supported by the disassembler IDA.
INF	INFScript	Malware for one of the Windows INF (installer) script interpreters. We do not distinguish INF 'versions' or 'type' in the platform name.
IRC	mIRCScript	Malware for the mIRC script interpreter.
Java	Java	Malware for some version of the Java runtime environment (standalone or browser–embedded).
JS	JavaScript	Malware for the Jscript and/or JavaScript interpreter. Hosting does not affect the platform designator. Standalone JS malware that requires MS JS under WSH, HTML–embedded JS malware, and JS malware embedded in Windows compiled HTML help files (.CHM), all fall under this platform type.
Linux	Linux	Malware specific to the Linux platforms and others closely based on it. *Unix* is still the preferred platform name.
LM	LotusMacro	Macro malware for Lotus 1–2–3.
MacOS	MacOS	Malware for the Macintosh OS prior to OS X.
MeOS	MenuetOS	Malware for the Menuet operating system.
MPB	MapBasic	Malware written in MapBasic.
MSIL	MSIL	Malware that requires a Microsoft Intermediate Language interpreter platform.
Mul	Multi	This is a pseudo–platform used for multi–platform malware.
O97M	Office97Macro	This is a pseudo–platform name reserved for macro

		malware that infects across at least two applications within the Office 97 and later suites. The newer, more generic pseudo–platform name Mul (or Multi in long–form) is preferred for such cases.
OneC	OneCScript	Malware for the Russian accounting package 1C.
OS2	OS2	Malware for OS/2.
OSX	OSX	Malware for Macintosh OS X or a subsequent, essentially similar, version. Use Unix instead whenever possible.
P98M	Project98Macro	Macro malware for VBA for Project, that shipped in Project 97 and later.
PalmOS	PalmOS	Malware for PalmOS (any version).
Perl	Perl	Malware that requires a Perl interpreter. Hosting does not affect the platform designator—standalone Perl infectors under Unix(–like) shells, ones that require Perl under WSH and HTML–embedded Perl malware all fall under this platform type.
PHP	PHPScript	Malware that requires a PHP script interpreter.
PIRC	PirchScript	Malware for the Pirch script interpreter.
PP97M	PowerPoint97Macro	Macro malware for VBA for PowerPoint, that shipped with Office 97 and later.
ProScript	ProScript	Malware written in ProScript.
PS	PostScript	Malware that requires a PostScript interpreter.
PSPOS	PSPOS	Malware for the Sony PlayStation Portable platform.
PU97M	Publisher97Macro	Macro malware for VBA for Publisher 97 and later.
Py	PythonScript	Malware written in the language Python.
REG	Registry	Malware that requires a Windows registry file (.REG) interpreter (we do not distinguish .REG versions or ASCII vs. Unicode).
Ruby	RubyScript	Malware for Ruby.
SH	ShellScript	Malware that requires a Unix(–like) shell script interpreter. Hosting does not affect the platform name. Shell malware specific to Linux, Solaris, HP–UX or other Unices, or specific to csh, ksh, bash, tcsh or other interpreters all fall under this platform name.
Solaris	Solaris	For Solaris–specific malware. *Unix* is still the preferred platform name.
SymbOS	SymbianOS	Malware for the Symbian (EPOC6 and above) OS.

TIOS	TIOS	Malware written for the Texas Instruments range of programming calculators (e.g., TI-89).
Unix	Unix	This is the preferred platform name for binary (ELF, COFF or a.out format) malware on Unix platforms. For shell script malware, see the SH name.
V5M	Visio5Macro	Macro malware for VBA for Visio, that shipped in Visio 5.0 and later.
VBS	VBScript	Malware for the Visual Basic Script interpreter. Hosting does not affect the platform designator. Standalone VBS infectors that require VBS under WSH, HTML–embedded VBS malware, and malware embedded in Windows compiled HTML help files (.CHM), all fall under this platform type.
W16	Win16	Malware for one of the 16–bit Windows *x*86 OSes.
W2M	Word2Macro	Macro malware for the WordBasic interpreter included in Microsoft Word 2.0.
W32	Win32	Malware for one of the 'true' 32–bit Windows *x*86 OSes (i.e. not Win32s, not CE, but Windows 9*x*, ME, NT, 2000, XP on *x*86).
W64	Win64	Malware for the 64-bit versions of Windows.
W97M	Word97Macro	Macro malware for VBA for Word (that shipped in Word 97) and/or later. Changes in VBA between Word 97 and 2002 versions inclusive are sufficiently slight that we do not distinguish platforms even if the malware makes a version check or uses one of the few VBA features added in the later VBA versions.
WBS	WinBATScript	Malware written for the Wilson WindowWare WinBatch interpreter.
WCE	WinCE	Malware for the PocketPC platform.
WHS	WinHexScript	Malware written in the scripting language supported by the binary editor WinHex.
WM	WordMacro	Macro malware for WordBasic as included in Microsoft Word 6.0, Word 7.0, Word 95 and Word for Macintosh 5.*x*.
WPM	WordProMacro	Macro malware for the Lotus WordPro word processor.
X97M	Excel97Macro	Macro malware for VBA for Excel, that shipped in Excel 97 and later.
XF	ExcelFormula	Malware written in the Excel Formula language that has shipped in Excel since the very early days.

XM	ExcelMacro	Macro malware for VBA that shipped in Microsoft Excel 5.0.