Evolvable Malware

Sadia Noreen[†], Shafaq Murtaza[†], M. Zubair Shafiq[‡], Muddassar Farooq[†,‡]
[‡]Next Generation Intelligent Networks Research Center (nexGIN RC)
[†]National University of Computer & Emerging Sciences (FAST-NUCES)
Islamabad, 44000, Pakistan
{sadia.noreen,shafaq.murtaza}@nu.edu.pk,{zubair.shafiq,muddassar.farooq}@nexginrc.org

ABSTRACT

The concept of *artificial evolution* has been applied to numerous real world applications in different domains. In this paper, we use this concept in the domain of virology to evolve computer viruses. We call this domain as "Evolvable Malware". To this end, we propose an evolutionary framework that consists of three modules: (1) a code analyzer that generates a high-level genotype representation of a virus from its machine code, (2) a genetic algorithm that uses the standard selection, cross-over and mutation operators to evolve viruses, and (3) the code generator converts the genotype of a newly evolved virus to its machine-level code. In this paper, we validate the notion of evolution in viruses on a well-known virus family, called `Bagle`. The results of our proof-of-concept study show that we have successfully evolved new viruses–previously unknown and known-variants of `Bagle`–starting from a random population of individuals. To the best of our knowledge, this is the first empirical work on evolution of computer viruses. In future, we want to improve this proof-of-concept framework into a full-blown virus evolution engine.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Invasive Software

General Terms

Experimentation, Security

Keywords

Artificial Evolution, Computer Virus, Genetic Algorithm

1. EVOLVABLE MALWARE: MYTH OR REALITY

Computer Malware or simply put *malware* is a software or program that damages the computer system or does something unwanted to the computer [1]. Technically, *malware* is a broader term that includes viruses, worms, backdoors, exploits, etc. However, the most well-known type of malware is *virus*; the term which was coined by Fred Cohen in 1983[1] [7]. The fundamental reason of creating such nuisances is mostly criminal instinct of bad guys who want to create chaos or panic in the society–sometimes with ulterior financial motives. In fact, since the dawn of the new millennium, computer malware creation has emerged as a commercial industry with revenues skyrocketing to several million dollars [6].

Computer malware have been a major threat to the computer systems and networks since 1990s. However, the malware sophistication has significantly improved since the early days. The advancement in functionality and behavior of computer malware can be categorized into five distinct generations[2] [17]. The first generation malware were quite simple, i.e., they caused infection by simply attaching themselves to the code sections of *benign* executables. The malware in second generation had some additional functionality such as *self-recognition*. The malware of third generation have *stealth* capabilities that makes them difficult to detect. The malware belonging to the fourth generation use *armoring* techniques to protect themselves. Finally, the malware of the current generation use *polymorphic* techniques to obfuscate their code with every replication.

The reader should appreciate that the development of state-of-the-art malware is a fairly laborious task and it usually takes years of experience for a malware writer to master this art or science, whichever way you may call it. Moreover, luck also plays its part in the malware writing process. Keeping this in mind, malware writers have recently developed "malware creation engines", which create different variants of a given malware; mostly by applying compression techniques or by modifying the code section using *garbage insertion* or *instruction substitution*. However, such techniques are mostly naïve, and the developed variants essentially have the same functionality and semantics.

Some people think that *polymorphic* viruses are an example of the 'malware evolution process'. However, we believe that it is not evolution in the true sense. All polymorphic viruses–as of today–only provide a change in the code structure but their functionality remains the same [11]. We here

GECCO'09, July 8–12, 2009, Montréal Québec, Canada.

[1]Throughout this paper, the terms *malware* and *virus* are used interchangeably.
[2]This classification is not based on the chronological order but on their characteristics; like how sophisticated a malware is in its behavior and how much destruction it can cause.

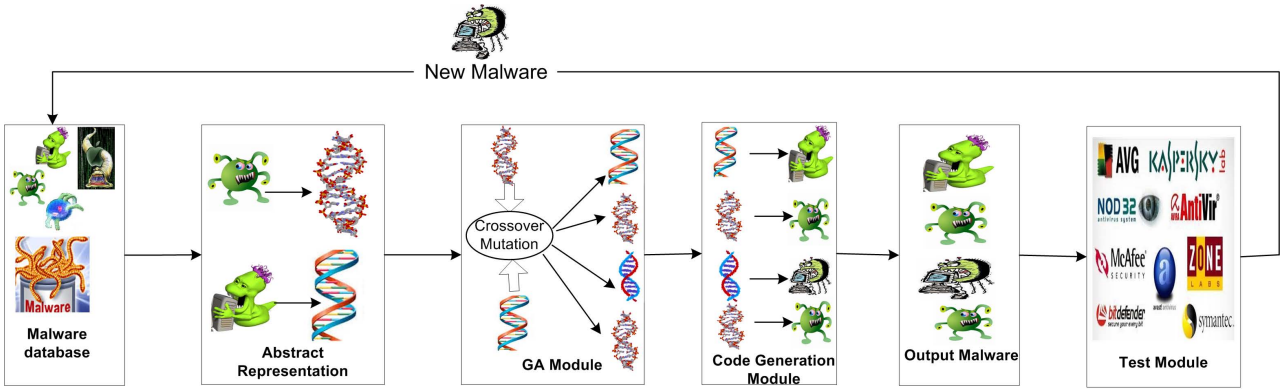


Figure 1: Architecture of the proposed malware evolution framework

formally define malware evolution as *a process which brings a change in the behavior, functionality or semantics of a given malware*, rather than a mere change in its structure. Keeping in mind the above-mentioned definition of malware evolution, many virology experts consider that 'true evolution' in computer malware is still a myth.

2. EVOLVABLE MALWARE FRAMEWORK

In our research, we propose a framework for evolving computer viruses. The architecture of the proposed malware evolution framework is shown in Figure 1. Our framework evolves new computer malware from a database of known malware.

The first step in malware evolution is the high-level abstract representation (or genotype) of a given malware. The development of high-level representation requires significant understanding of the functionality and structure of the malware. This representation determines the 'quality of evolution' achieved by the proposed framework. A sophisticated representation would include subtle functional details such as the list of victim applications, the list of ports used for propagation and the details of registry entries. The first module of our framework extracts functional details of the malware.

The second step is the application of evolutionary algorithms to the high-level representation. In our study, we have used the standard *genetic algorithm* (GA) for this purpose. GA is an evolutionary algorithm that puts a great deal of emphasis on selection, recombination and mutation acting on a genotype that is decoded and evaluated for fitness [21]. Several types of selection, recombination and mutation algorithms are available to choose from. In our study, we have carried out a comprehensive evaluation of several well-known selection and recombination algorithms. We have also tried to determine the optimal combination of selection and recombination procedures for our malware evolution process.

After the application of GA, new individuals in the population are translated back from high-level representation to machine-level code. It is effectively a code generator that converts an abstract representation to a machine-level code. The other module of our proposed framework translates the high-level representation back to the machine-level code.

Finally, the generated virus files are tested using commercial antivirus software to check if they are known variants of the given malware. Even if the generated malware files are not detected by any antivirus software, it is possible that it is an unknown working variant of the given malware[3]. To establish that the newly generated unknown individual is a malware, we have to execute it in a real operating system or a virtual machine. In order to make our current study plausible, we have restricted the scope of our study to the generation of known variants of a virus only.

3. PROOF-OF-CONCEPT STUDY USING BAGLE

In this section, we present the details of our proof-of-concept study using a well-known virus family called `Bagle`. We first provide the functional details of `Bagle` that will help a reader to understand its high-level genotype representation and the process of generating it from the machine-level code.

3.1 With Microscope and Tweezers: Analysis of Bagle family

It is desirable that the genes in the high-level genotype representation are common to all variants of the `Bagle` family. To develop a suitable high-level representation, we have taken the fingerprints of different variants of `Bagle` family and closely analyzed their different characteristics such as their parameters, their functional flow and the specific functions performed by each variant [1], [5], [3]. We have used a well-known disassembler and debugger, called *IDA Pro*, to perform this forensic analysis [2].

`Bagle`, also known as *beagle*, is a mass-mailing computer worm that infects all versions of the Microsoft Windows. The first strain `Bagle.a` was not a big success as it did not propagate widely. However, its next variation i.e. `Bagle.b` is far more infectious. It gathers email addresses from a victim's computer and emails itself as an attachment to these recipients. It actually has its own SMTP engine to mass-mail itself. Moreover, it also opens a backdoor channel on a TCP port and copies itself to the 'system directory'.

All variants of the `Bagle` family are backdoors and we have taken 15 of them for our research. The results of our forensic analysis have revealed that the major differences among different variants of the family are:

[3]Most of the commercial antivirus software are signature based, therefore, they might not have the signature of new malware in their signature database.

- The date prior to which they continue their destructive activities.
- The port numbers they listen to, for example `Bagle.a` uses the port 6777 where as `Bagle.b` uses the port number 8866.
- The packers that are used for packing or doing encryption.
- The applications they execute to conceal themselves from a user's suspicion.
- Additional functionalities like killing the antivirus processes and peer-to-peer propagation.

```
beagle: 00401683  mov  word ptr[ebp+var_20], 7D4h
beagle: 00401689  mov  word ptr[ebp+var_20+2], 1
beagle: 0040168F  mov  word ptr[ebp+var_1A], 1Ch
```

Figure 2: Activation date of `Bagle.a`

Figure 2 shows the piece of code that holds the date information of `Bagle.a` in the hex format. This date is equivalent to $28th$ January, 2004. The worm first gets the current date from the system and compares it with the date stored in the virus file [16]. If it is prior to $28th$ January, 2004 then the worm is executed; otherwise it is terminated.

```
au: 00401683   mov   word ptr[ebp+var_20], 7D4h
au: 00401689   mov   word ptr[ebp+var_20+2], 2
au: 0040168F   mov   word ptr[ebp+var_1A], 19h
```

Figure 3: Activation date of `Bagle.b`

`Bagle.b` does the same thing but it has a different date stamp. It checks for the date entry of $25th$ February, 2004 as shown in the Figure 3. Some variants of `Bagle` have a date stamp present in their code but this feature is absent in several variants.

`Bagle` also creates the registry entries. The two registry entries *uid* and *frun* are created in HKEY_CURRENT_USER\SOFTWARE\Windows98. The presence of *frun* in the registry means that virus has run at least once on this machine. After this entry, `Bagle` creates another entry *d3dupdate.exe* in HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run. The objective of this entry is to make sure that the code automatically starts next time the Windows is started. All these entries are declared in the data section of the virus file as shown in Figure 4.

The data section in Figure 4 has an entry titled *bbeagle.exe*; it is the process name of the virus that will be displayed in the task manager. Another function that `Bagle` performs to conceal is that its different variants use different applications. For example, `Bagle.a` conceals itself into *calc.exe* as shown in Figure 4. Similarly, `Bagle.b` conceals itself into *sndrec32.exe* and `Bagle.c` uses *notepad.exe*.

```
.data:004056E1  ; char CmdLine[]
.data:004056E1    CmdLine       db 'calc.exe',0
.data:004056EA    aOpen         db 'open',0
.data:004056EF  ; char SubKey[]
.data:004056EF    SubKey        db 'SOFTWARE\
                  Windows98',0 ;
.data:00405702  ; char ValueName[]
.data:00405702    ValueName     db 'uid',0
.data:00405706  ; char aSoftwareMicros[]
.data:00405706    aSoftwareMicros db
                  'SOFTWARE\Microsoft\
                   Windows\CurrentVersion\Run',0
.data:00405734  ; char aD3dupdate_exe[]
.data:00405734    aD3dupdate_exe  db
'd3dupdate.exe',0   .data:00405742   ; char
aBbeagle_exe[]
.data:00405742    aBbeagle_exe  db '\bbeagle.exe',0
.data:0040574F  ; char aFrun[]
.data:0040574F aFrun          db 'frun',0
```

Figure 4: Application used by `Bagle` to conceal itself, and registry entries

```
.data:00405003   dword_405003    dd 1A79h
.data:00405007   ;char String1[]
.data:00405007   String1  db '151.201.0.39', 0
```

Figure 5: Port Number and DNS for `Bagle.a`

Figure 5 shows the code section where hard coded DNS and port numbers for `Bagle.a` are declared. It is evident in Figure 5 that the port number for `Bagle.a` is $1A79h$ i.e. 6777 and the IP address of the DNS server is 151.201.0.39. Moreover, `Bagle` also collects the email addresses from a victim's computer and for this purpose it searches for the files with the extensions *.wab, .txt, .htm, .html* and avoids the domains such as *@hotmail.com, @msn.com, @microsoft, @avp* as shown in Figure 6. The malware contains an email subject, email body and contents for these fields are also defined in the data section of the malware. For `Bagle.a`, the email subject is *Hi* and email body contains the string *Test*. The attachment name is randomly selected by `Bagle.a`. The extension of attachment is *".exe"* which is mentioned in the data section. All these attributes are common in most of the variants of the `Bagle` class and will be used to develop the high-level abstract representation.

Figure 7 shows the list of websites to which `Bagle.a` informs about the successful infection. The list of websites is usually hard-coded in the data segment of the malware. Virus writers use this functionality to obtain a feedback about the propagation statistics of the malware. However, this tactic can also backfire as the list of websites can be used to trace back the identity of the malware's author.

3.2 Abstract Representation of Bagle family

To apply GA, we first need to formulate the high-level abstract genotype representation of the `Bagle` virus; more commonly known as the chromosome structure. A good

Table 1: Abstract Representation of `Bagle`

Feature	Description	Examples
Date	The date checked by `Bagle` to (de-)activate its process	28 January, 2004
Application	The applications used to conceal `Bagle`	calc.exe, notepad.exe, sndrec32.exe
Port Number	Port opened by `Bagle` to send or receive commands	2475, 6777, 2556
Attachment	Name of the attachment used by `Bagle`	Random characters
Attachment Extension	It specifies the extension of the attachment	.rar, .exe, .pif, .zip
Websites	`Bagle` contact the websites to inform about the infection	http://www.it-msc.de/1.php, http://www.getyourfree.net/1.php
Domain	`Bagle` ignores to email itself to the domains specified	@hotmail.com, @msn.com
Email Body	Contains the email body of `Bagle`	Test=), YoursID<Random Characters>
Email Subject	Specifies the subject of the email	Hi, Subject:ID <Random Characters>
Registry Variable	Contains the name of registry variables used by `Bagle`	au.exe, d3dupdate.exe
Virus Name	Name of the `Bagle` shown in the task manager	bbeagle.exe, au.exe, readme.exe
File Extension	File extensions to be searched in the fixed directories	.wab, .txt, .htm, .php
Process Terminated	Processes terminated by `Bagle`	atupdater.exe, aupdate.exe
P2P Propagation	Names used by `Bagle` to copy itself to peer computers	ACDSee 9.exe, Ahead Nero 7.exe

```
.data:0040501C a_wab          db '.wab',0
.data:00405021 a_txt          db '.txt',0
.data:00405026 a_htm          db '.htm',0
.data:0040502B a_html         db '.html',0
.data:00405032 ; char Srch[]
.data:00405032 Srch           db '.r1',0
.data:00405036 a@hotmail_com  db
                              '@hotmail.com',0
.data:00405043 a@msn_com      db
                              '@msn.com',0
.data:0040504C a@microsoft    db
                              '@microsoft',0
.data:00405057 a@avp_         db '@avp.',0
```

Figure 6: File extensions to be searched and domains to be ignored by `Bagle.a`

```
.data:0040519C aHttpVipweb_ru1  db 'http://
               vipweb.ru/1.php',0
.data:004051B3 aHttpAntolCo_ru  db 'http://
               antol-co.ru/1.php',0
.data:004051CC aHttpWww_bagsDo db
               'http://www.bags-
               dostavka.mags.ru/1.php',0
.data:004051F3 aHttpWww_5x12_r db
               'http://www.5x12.ru/1.php',0
.data:0040520C aHttpBoseAudio_ db 'http://
               bose-audio.net/1.php',0
.data:00405228 aHttpWww_sttngd db
               'http://www.sttngdata.de/
               1.php',0
.data:00405246 aHttpWh9_tuDres db 'http://
               wh9.tu-dresden.de/1.php',0
```

Figure 7: List of websites to inform about infection

representation of a chromosome generally covers majority of the features of a given virus. Consequently, it helps us to map each variant of `Bagle` to its genotype representation and vice versa.

Some common attributes of `Bagle` class have been discussed in the previous subsection. All of these attributes are also present in the other variants of `Bagle` class with different values. For example, the date stamp attribute is present in both `Bagle.a` and `Bagle.b` but with different dates. Likewise, the name of application used to conceal and the port number are also common attributes of `Bagle` class. Therefore, *date*, *application* and *port number* constitute the first three members of the chromosome representation (see Table 1).

The next gene of the chromosome is the *email attachment*; malware sends messages with an attachment which is chosen randomly for some variants. Moreover, the extension of this attachment is another member of representation. The extensions can be *.exe, .zip, .src, .rar, .pif* and is mentioned in the data section of the executable file of the malware. The attribute *websites* contains the list of websites and the malware contacts these websites with details of the infection. The list of these websites is extracted from the data segment of the malware. Modern malware mostly access some peer-to-peer network to get the list of websites.

The *domain* parameter of the chromosome representation scheme specifies the strings that are ignored by the `Bagle` i.e., the worm makes sure that the email address does not contain the strings specified in the domain parameter. *Mail Body* is another parameter extracted from the code section that contains the message. *Email Subject* is another relevant parameter.

The *Registry Variable* parameter makes sure that the malware automatically initiates itself at system startup. This parameter has different values for different variants of `Bagle` class. The next parameter is *virus name*. It specifies the name of the process initiated by malware as shown in the task manager. Moreover, the worm also searches hard disks to locate files with the extensions like *.wab* and *.txt* in order to collect the email addresses. The *File Extension* parameter holds the extensions of the files which it wants to search on hard disks.

Another important feature present in most of the variants of `Bagle` class is termination of several processes to disable well-known security software. This parameter is specified as *process terminated* in our chromosome representation. Finally we have peer-to-peer propagation which specifies the

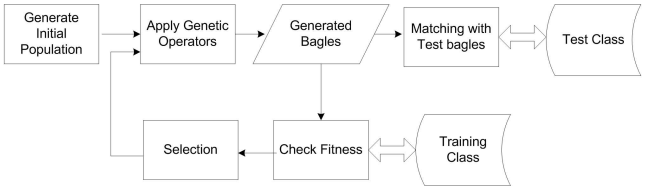


Figure 8: Elaborated experimental setup

names–used by `Bagle`–to copy itself to peer computers. The complete chromosome representation is summarized with examples in Table 1. Referring back to the definition of *malware evolution* presented in Section 1, a careful reader will notice that our abstract representation contains attributes that reflect the behavior and the functionality of `Bagle` family rather than mere code structure. For example, 'Application' feature determines the application used to conceal the malware.

4. EXPERIMENTS, RESULTS AND DISCUSSIONS

In this section, we provide the details of our experimental setup, results and discussions.

4.1 Experimental Setup

We have obtained 15 unique variants of `Bagle` from different malware sources such as *VX Heavens* [4] and *Offensive Computing* [3]. To formulate our test bed, we have divided `Bagle` samples into two categories: training and testing. The samples in the training category are utilized to guide the evaluation process: the fitness of the offsprings is a function of the similarity measure (will be shortly defined) of their chromosomes with that of training viruses. An exact match results in a fitness of 1. Moreover, once we have evolved new individuals that do not match with the training samples then we might arrive at three conclusions: (1) the new individual is a malware in the testing category, (2) the new individual is an unknown `Bagle` virus, and (3) the new individual is not a `Bagle` virus. We have already mentioned that the facts 2 and 3 can only be established once we execute the virus in real operating system or virtual machine. We argue that if we have generated testing `Bagle` viruses from a random population of individuals, then effectively we have achieved evolution because testing viruses are never used in guiding the evolution process. Fact 1, therefore, helps us in establishing the evolution claim in a time efficient fashion.

Figure 8 shows our experimental setup. For all features in abstract representation, we have developed a *gene library*. For example, the gene library for *port number* contains all possible values that can be used by the malware of `Bagle` family. Similarly for the *date* feature of the abstract representation, the gene library contains all possible date stamps that can be used by `Bagle`.

As a first step, our implementation generates an initial population from the gene library. Then well-known genetic operators–such as crossover and mutation–are applied on the individuals of initial population. After applying GA operators, the genotypes of malware offsprings are obtained. The fitness (F) of offsprings is evaluated using the samples from training set. The fitness of an individual is directly proportional to its resemblance to the malware samples in the training set. The resemblance is calculated by comparing each parameter of the genotype to the respective genotype parameter of the training set. Moreover, the fitness is normalized by assigning a weight to every gene of the representation. For k genes, the fitness F is mathematically given by:

$$F = \sum_{i=1}^{k} \frac{f_i}{k}$$

Here f_i is determined by the resemblance of respective genes of an offspring and a sample virus in the training set. The generated offsprings are also compared to the testing set and the statistics are archived. The offsprings from the population are selected for next generation based on their fitness.

4.2 Results and Discussions

In our experiments, we have evaluated the performance of our prototype framework. We have carried out a comprehensive comparative study of different selection and crossover techniques. The selection methods used in our study are roulette wheel (R-Wheel), rank and tournament selections. The different crossover methods used in our study are one point (1-Pt), two point (2-Pt) and uniform crossovers. A tangential aim of this study is to evaluate the performance of our framework using different selection and crossover techniques for our domain. For this study, we have set the crossover probability at 0.75 and the mutation probability at 0.005. The population size of 500 individuals is used in our study.

We have fixed these parameters so that we can explore other interesting dimensions (such as the type of selection and crossover) of the design space. Here we emphasize that the prime focus of this study in not on finding the design point to evolve an individual with the highest fitness. We are not using GA to solve a typical optimization problem, rather we are using it as an evolution tool. Moreover, the traditional concept of fitness does not pertain for the malware evolution problem since there is no such malware as the fittest malware.

Tables 2 and 3 show the the means and the standard deviations of finesses obtained after 500 generations. The bold entries highlight the best results in every row. We are unable to find any correlation between the fitness of an offspring–once compared with the viruses in the training set–and the types of selection and crossover (see Table 2) techniques. For `Bagle.a` and `Bagle.c` and `Bagle.e`, rank selection with one-point crossover provides the best results. For `Bagle.b` roulette wheel selection with one-point crossover gives the best average fitness. For `Bagle.d` rank selection with two-point crossover provides the best fitness.

For fitness evaluated using the testing set, tournament selection with two-point crossover gives the best average fitness for `Bagle.f`, and `Bagle.i`. For `Bagle.j` rank selection with two-point crossover has provided the best average fitness. On the other hand, for `Bagle.k` and `Bagle.n` uniform crossover with rank selection has provided the best fitness. However, we cannot establish any pattern for best fitness with respect to the choice of selection and crossover technique.

To get more detailed insights, we also analyze the fitness of `Bagle.a` against training set for different selection and

Table 2: Fitness for training `Bagle` set using different selection and crossover methods. Bold values in every row highlight the best fitness.

Selection	Rank	Rank	Rank	R-Wheel	R-Wheel	R-Wheel	Tournament	Tournament	Tournament
Crossover	1-Pt.	2-Pt.	Uniform	1-Pt.	2-Pt.	Uniform	1-Pt.	2-Pt.	Uniform
Bagle.a	**0.951 ± 0.009**	0.948 ± 0.015	0.926 ± 0.034	0.924 ± 0.026	0.941 ± 0.110	0.918 ± 0.181	0.938 ± 0.133	0.900 ± 0.084	**0.850** ± 0.080
Bagle.b	0.897 ± 0.147	0.944 ± 0.015	0.935 ± 0.035	**0.957 ± 0.036**	0.931 ± 0.158	0.948 ± 0.007	0.930 ± 0.165	0.889 ± 0.160	0.846 ± 0.149
Bagle.c	**0.890 ± 0.141**	0.881 ± 0.007	0.887 ± 0.031	0.873 ± 0.002	0.882 ± 0.067	0.883 ± 0.103	0.886 ± 0.147	0.830 ± 0.088	0.806 ± 0.058
Bagle.d	0.912 ± 0.028	**0.928 ± 0.026**	0.915 ± 0.036	0.907 ± 0.011	0.901 ± 0.104	0.914 ± 0.075	0.910 ± 0.226	0.853 ± 0.094	0.835 ± 0.010
Bagle.e	**0.862 ± 0.113**	0.852 ± 0.014	0.842 ± 0.017	0.848 ± 0.009	0.852 ± 0.030	0.839 ± 0.075	0.846 ± 0.049	0.807 ± 0.073	0.785 ± 0.046

Table 3: Fitness for testing `Bagle` set using different selection and crossover methods. Bold values in every row highlight the best fitness.

Selection	Rank	Rank	Rank	R-Wheel	R-Wheel	R-Wheel	Tournament	Tournament	Tournament
Crossover	1-Pt.	2-Pt.	Uniform	1-Pt.	2-Pt.	Uniform	1-Pt.	2-Pt.	Uniform
Bagle.f	0.480 ± 0.007	0.499 ± 0.002	0.462 ± 0.023	0.495 ± 0.001	0.463 ± 0.035	0.461 ± 0.032	0.438 ± 0.014	**0.611 ± 0.021**	0.450 ± 0.0019
Bagle.i	0.483 ± 0.032	0.525 ± 0.013	0.524 ± 0.001	0.523 ± 0.009	0.516 ± 0.040	0.492 ± 0.021	0.502 ± 0.034	**0.559 ± 0.058**	0.530 ± 0.004
Bagle.j	0.486 ± 0.029	**0.525 ± 0.013**	0.524 ± 0.001	0.458 ± 0.015	0.507 ± 0.053	0.496 ± 0.007	0.502 ± 0.034	0.496 ± 0.040	0.500 ± 0.034
Bagle.k	0.461 ± 0.023	0.483 ± 0.027	**0.503 ± 0.001**	0.431 ± 0.023	0.479 ± 0.033	0.480 ± 0.012	0.492 ± 0.044	0.473± 0.033	0.484 ± 0.040
Bagle.n	0.447 ± 0.023	0.458 ± 0.028	**0.484 ± 0.002**	0.441 ± 0.001	0.477 ± 0.004	0.454 ± 0.022	0.417 ± 0.0021	0.443 ± 0.025	0.472 ± 0.0150

crossover techniques (see Figure 9). It is clear that the single point crossover technique eventually achieves the best fitness for all selection techniques. On the other hand, the convergence of two-point crossover method to the best fitness is fastest (i.e. in least number of generations) for all selection techniques.

Figure 9(a) shows that the solution converges more quickly in case of two-point crossover as compared to single-point and uniform crossover methods. Moreover, maximum fitness achieved by the single-point crossover is greater than the fitness achieved by the other two crossover methods. This trend is also prevalent for roulette wheel selection as shown in Figure 9(b) and rank selection (see Figure 9(c)).

We have carried out another useful analysis which provides answer to a relevant question: what is the relationship of the number `Bagle` samples in the evolved population with respect to different population sizes P. It is evident from Figure 10(a) that the percentage of offsprings that match to `Bagle` samples in the training set increase significantly as the population size increases. Remember that the training set is used to guide the evolution process. In comparison, this trend is not very clear in Figure 10(b). We again emphasize that the viruses in the testing set are *not* used to guide the evolution process. The percentage of offsprings that match to `Bagles` in the testing set oscillate about 50% for $P = 2000$. These are effectively the 'new' variants of `Bagle` that our malware evolution system has generated. Moreover, our system also generates 'unknown' variants of `Bagle` which we could not verify because of absence of the ground truth.

To verify our results, we have tested a sample of evolved malware with two commercial antivirus softwares: (1) AVG antivirus, and (2) Symantec antivirus. The results are shown in the Figure 11. AVG detects 98.98% of the evolved malware from training set and 1.02% of the evolved malware remain undetected (see Figure 11(a)). The statistics in Figure 11(b) shows that 14.46% of the evolved malware are detected with the name `W32.Sality.AE` which is not in the training or testing set. Further 26.64% of the evolved malware are detected as `W32.Beagle!gen` by Symantec antivirus and 58.9% remain undetected. *These results prove our claim that the proof-of-concept malware evolution engine has successfully evolved unseen malware.*

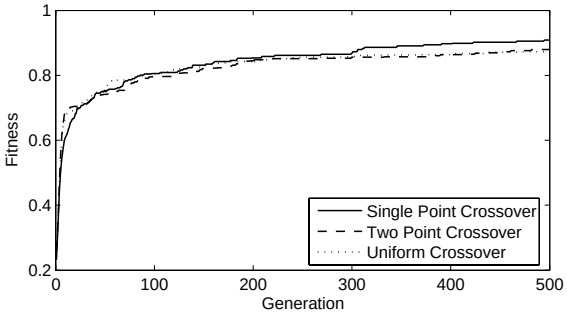
5. REAL WORLD APPLICATIONS

In this section, we provide some real world applications of our proposed *Evolvable Malware* framework. We describe its applications separately below:

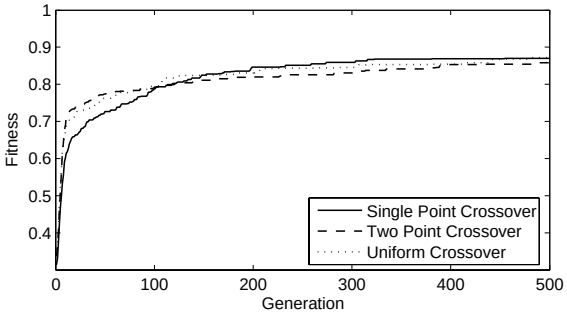
- Though in this paper our proposed framework is posed as an evolvable malware system, however, it is in fact an example of 'software evolution'. Software evolution is a phenomena associated with modifying existing software systems [12]. A possible advantage of evolving software is that it emphasizes reusability of components instead of developing them from scratch.

- Our proposed evolving malware framework can be used to test host-based antivirus software, especially all non-signature based antivirus products. (Developing non-signature based techniques for viruses is still an active area of research.) A robust non-signature based antivirus software will detect the majority of the evolved malware variants.

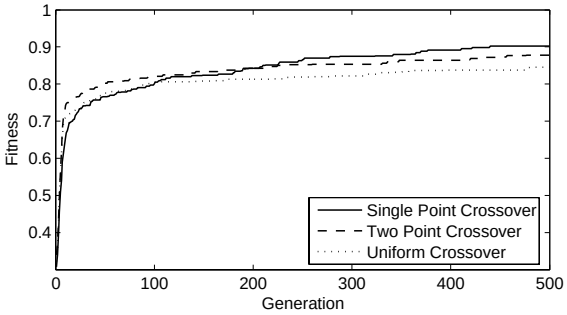
- A wide variety of network based malware–commonly known as worms–can also be evolved by our proposed framework. These evolved variants can be of great value in network security research.



(a) Tournament selection

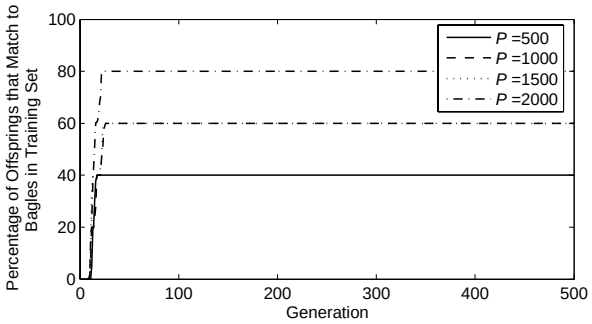


(b) Roulette Wheel selection

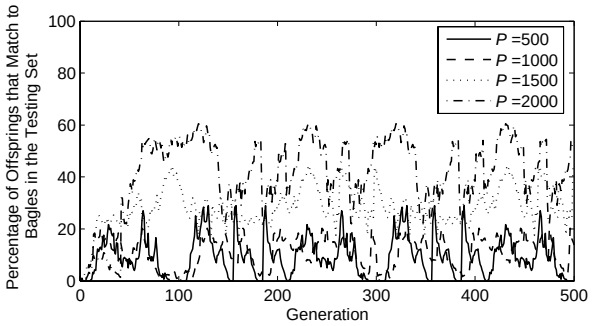


(c) Rank selection

Figure 9: Fitness plots for different selection and crossover methods



(a) Percentage Matches to the Training Set



(b) Percentage Matches to the Testing Set

Figure 10: Percentage of offsprings that match to `Bagle` samples in the training and testing set

6. RELATED WORK

To the best of our knowledge, artificial evolution is applied for the first time in evolving new computer malware. The concept of artificial life for computer viruses has been proposed in the past [11]. In [17], the authors carried out a study to verify different properties of life such as self-reproduction, metabolism and growth, in computer malware. The authors concluded that some properties are present to an extent in existing computer viruses. In [11], the author proposed the idea of "Darwinian Genetic Mutation Engine" (DGME). According to [11], DGME will have the ability to replicate and mutate the computer viruses. This idea still seems very futuristic and very little work has been done to evaluate it. We believe that our proposed framework in this paper is the first practical implementation and demonstration of the exciting domain of *Evolvable Malware*.

Apart from evolving computer viruses and software, evolutionary techniques have been extensively applied in a variety of fields to provide solutions to different real world problems. To maintain focus, we will focus on applications in which intelligent software behaviors are evolved. In the field of artificial intelligence, GA has been used in [19] for computer gaming, e.g., in chess playing GA is used to optimize the evaluation function to replicate the behavior of an intelligent entity or mentor. GA is used to tune the parameters of evaluation function to mimic the behavior of a mentor.

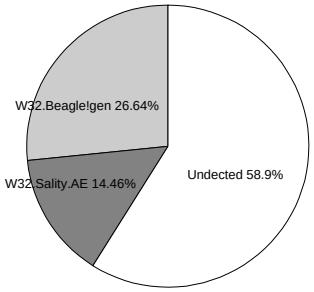
In [9], evolutionary programming is used to automatically programme a robot. In fact, the behavior of robot is evolved to perform a given task without explicitly programming it. In another seminal work, evolutionary computation is used to evolve circuits [10]. In [8], GA is used in the field of music composing and music mixing. This technology has already been incorporated into new software. In [20], collaboration between human musicians and robot musicians utilizes genetic algorithm to produce new and inspiring music.

GA has been successfully used in network security problems– for example in network intrusion detection systems. In [18], the authors have successfully deployed GA to extract a subset of traffic features to increase the detection rate. Moreover, GA has also been used for dynamic network redesigning with reconfigurable links in case of a major network failure [15]. Using GA, the techniques for jamming radars have also been developed [14]. GA is also applied to web mining– an application of data mining–to increase the coverage of a web search engine [13].

7. CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed an evolvable malware

(a) Evolved `Bagles` scanned with AVG



(b) Evolved `Bagles` scanned with Symantec

Figure 11: Results obtained by scanning the evolved malware by antivirus software

framework. The basic idea is to use evolutionary computation to evolve computer viruses. For proof-of-concept, we have used this framework to evolve new variants of `Bagle`. To this end, we have developed a high-level feature representation of `Bagle` family and applied GA to the developed chromosome representation. Our experiments show that starting from a few training samples we have successfully evolved new and unknown variants of `Balge` family. We have verified our results using real variants of `Bagle` family that were not used to guide the evolution process of our framework.

In future, we can make our implementation more sophisticated by further developing the code analysis and code generation modules. A full-blown version of our proposed framework can be generalized for all types of malware.

8. REFERENCES

[1] F-Secure Virus Description Database, available at `http://www.f-secure.com/v-descs/`.

[2] The IDA pro disassembler and debugger, available at `http://www.hex-rays.com/idapro/`.

[3] Offensive Computing, available at `http://www.offensivecomputing.net`.

[4] VX Heavens Virus Collection, VX Heavens website, available at `http://hvx.netlux.org`.

[5] Kaspersky Lab, VirusList.Com, available at `http://www.viruslist.com/en/viruses/encyclopedia/`.

[6] J.M. Bauer, J.G. Michel and Y. Wu. "ITU Study on the Financial Aspects of Network Security: Malware and Spam", ICT Applications and Cybersecurity Division, International Telecommunication Union, Final Report, July 2008, available at `http://www.itu.int/ITU-D/cyb/cybersecurity/docs/itu-study-financial-aspects-of-malware-and-spam.pdf`.

[7] F. Cohen, "Computer Viruses", PhD thesis, University of Southern California, 1985.

[8] G. Gabrani, P. Bhargava, B. Bhawana and G.S. Gill. "Use of Genetic Algorithms for Indian Music Mixing", ACM Ubiquity, 9(10), Article 1, ACM Press, 2008.

[9] J.R. Koza, F.H.Bennett, D. Andre and M.A. Keane "Reuse, parameterized reuse, and hierarchical reuse of substructures in evolving electrical circuits using genetic programming", International Conference on Evolvable Systems: From Biology to Hardware, Volume 1259 of Lecture Notes in Computer Science, pp. 312-326, Springer, UK, 1996.

[10] J.R. Koza and J.P. Rice, "Automatic Programming of Robots using Genetic Programming" 10th National Conference on Artificial Intelligence, pp. 194-201, Association for the Advancement of Artificial Intelligence (AAAI), 1992.

[11] M.A. Ludwing, "Computer Viruses, Artificial Life and Evolution", American Eagle Publications, 1993.

[12] J.Gray, R. Klefstad, "Adaptive and Evolvable Software Systems: Techniques, Tools, and Applications", 38th Annual Hawaii International Conference on System Sciences (HICSS), page 274, IEEE Press, 2005.

[13] M.H. Marghny and A.F. Ali, "Web Mining based on Genetic Algorithm", IGCST International Journal on Artificial Intelligence and Machine Learning, Special Issue on AI Classification & Analysis Techniques, 2006.

[14] H.J.F. Moen and S. Kristoffersen, "Multi-resistant radar jamming using genetic algorithms", Genetic and Evolutionary Computation Conference (GECCO), pp. 1595-1602, ACM Press, USA, 2008.

[15] D. Montana, T. Hussain and T. Saxena, "Adaptive Reconfiguration Of Data Networks Using Genetic Algorithms", Genetic and Evolutionary Computation Conference (GECCO), pp. 1141-1149, ACM Press, USA, 2002.

[16] K. Rozinov, "Reverse code engineering: An In-depth Analysis of the Bagle Virus", 6th Annual IEEE SMC Information Assurance Workshop (IAW), pp. 380-387, IEEE Press, USA, 2005.

[17] E.H. Spafford, "Computer viruses as Artificial Life", Journal of Artificial Life, 1(3), pp. 249-265, MIT Press, 1994.

[18] G. Stein, B. Chen, A.S.Wu and K.A. Hua, "Decision tree classifier for Network Intrusion Detection with GA-based Feature Selection", 43rd Annual ACM Southeast Regional Conference, pp. 136-141, USA, 2005.

[19] O.D.- Tabibi, M. Koppel and N.S. Netanyahu, "Genetic algorithms for mentor-assisted evaluation function optimization", Genetic and Evolutionary Computation Conference (GECCO), pp. 1469-1476, ACM Press, USA, 2008.

[20] G. Weinberg, M. Godfrey, A. Rae and J. Rhoads, "A Real-time Genetic Algorithm in Human-robot Musical Improvisation", 4th International Symposium on Computer Music Modeling and Retrieval, Sense of Sounds, Volume 4969 of Lecture Notes in Computer Science, pp. 351-359, Springer, 2008.

[21] D. Whitley, "An Overview of Evolutionary Algorithms: Practical Issues and Common Pitfalls", Information and Software Technology, 43(14), pp. 817-831, 2001.