Experimental Investigation in the Impact on Security of the Release Order of Defensive Algorithms

Suliman A. Alsuhibany, Ahmad Alonaizi, Charles Morisset,
Chris Smith, and Aad van Moorsel

Centre for Cybercrime and Computer Security
Newcastle University, UK, School of Computing Science
Claremont Tower, NE1 7RU, United Kingdom
{suliman.alsuhibany,a.alonaizi,charles.morisset,
aad.vanmoorsel}@ncl.ac.uk

Abstract. In the practical use of security mechanisms such as CAPTCHAs and spam filters, attackers and defenders exchange 'victories,' each celebrating (temporary) success in breaking and defending. While most of security mechanisms rely on a single algorithm as a defense mechanism, we propose an approach based on a set of algorithms as a defense mechanism. When studying sets of algorithms various issues arise about how to construct the algorithms and in which order or in which combination to release them. In this paper, we consider the question of whether the order in which a set of defensive algorithms is released has a significant impact on the time taken by attackers to break the combined set of algorithms. The rationale behind our approach is that attackers learn from their attempts, and that the release schedule of defensive mechanisms can be adjusted so as to impair that learning process. This paper introduces this problem. We show that our hypothesis holds for an experiment using several simplified but representative spam filter algorithms—that is, the order in which spam filters are released has a statistically significant impact on the time attackers take to break all algorithms.

Keywords: Release Order of Defensive Algorithms, learning, Experimentation, Security and Protection.

1 Introduction

Many security solutions are based on algorithms that aim to protect system resources from misuse. These algorithms encode a set of rules that aim to characterize and recognize attempts of misuse, and prevent any adverse effect on system resources. Examples of such algorithms include spam-filters, CAPTCHAs and Anti-Phishing solutions. As attackers interact with the system, they receive feedback that augments their knowledge of the rules used by the system to characterize misuse. Accordingly, they are able to adapt their future interactions in accordance with this augmented knowledge, increasing their ability to break the defensive algorithms, until eventually reaching the point where the security mechanism is broken: the spam-filter rules are overridden, the CAPTCHAs are automatically deciphered, etc.

A. Cuzzocrea et al. (Eds.): CD-ARES 2013 Workshops, LNCS 8128, pp. 321–336, 2013.

Once a mechanism is broken, the security officer must deploy another one, which will eventually be in turn broken, leading the officer to deploy a new mechanism, and so on and so forth [7]. Deploying a defensive mechanism has a cost, for example, the cost of deploying a SPAM filter within a particular organization has been calculated at about fifteen thousand euro for the first year [22]. Furthermore, the security officer must usually work within a given budget constraint, and has therefore a limited number of defense mechanisms to deploy. This raises the question whether we can better plan the release of defensive algorithms, so as to extend the period for which the combined set of algorithms is effective? For instance, could it be useful to break up one defensive algorithm into multiple algorithms, and release them one by one? Or could it be useful to reorder the release of the various algorithms to maximize the overall time taken by the attacker to break all algorithms?

In this paper, we are interested to address the following research question: *"Does the order in which different defensive mechanisms are released impact the time an attacker needs to break each one of them?"* The main reasoning behind considering this question may be answered affirmatively is that the time it takes an attacker to break a defensive algorithms may depend on what the attacker has learned from earlier successful attacks on similar algorithms. If that is true, one may be able to defend better against attacks by impairing the process of learning of attackers. There may be many ways in which one can try to achieve this, but in this paper we consider the most direct implication of this reasoning, namely that the order in which defensive algorithms are released may impact the learning process.

We experimentally test the hypothesis that the release order matters. We conduct an experiment with simplified but representative spam filter algorithms, asking two groups of twenty subjects to break these algorithms. The algorithms are presented in a reversed order to the respective groups. By analyzing these groups, the order of release can have a statistically significance influence in the time required by an attacker to break a mechanism. In other words, there is potential to improve the security of a system by optimizing the order in which the defensive algorithms are released.

The work presented here contributes to the effort captured recently in the term 'science of security', to emphasize scientific and empirical studies for security mechanisms rather than designing a set of best practices. In this regard, an important contribution of this work is the method and protocol used to answer the research question.

In addition, the effect of presentation orders on the learning mechanism is not new. Previous researches in the field of education and psychology provide several insights into the effect of presentation orders [15, 16, 17, 20]. However, to the best of our knowledge, we are the first to address this particular issue of the release order strategy. There exists a considerable amount of related work considering the attack and defense interaction as a game-theoretic problem, but these formulations do not fit exactly with our approach. We refer to Section 6 for a further discussion.

The rest of this paper is organized as follows. Section 2 describes the problem as precise as possible through a detailed system model. Section 3 describes the experimental study, and its results are presented in Section 4. The discussion is presented in Section 5. Section 6 discusses related work. Finally, we conclude with overall discussions and future work in Section 7.

2 Problem Definition

In order to precisely define the problem under consideration, we start by providing an abstract system model of the attack scenario, involving the attacker and the system as shown in Figure 1. It describes a general class of security solutions which include CAPTCHAs, certain spam filters, intrusion tolerance algorithms, etc. This class of mechanisms is characterized by an intelligent defensive algorithm being attacked and eventually broken, and then being replaced by a new intelligent defensive mechanism, etc. Furthermore, this model provides the basis of the experimental study that we present in Section 3.

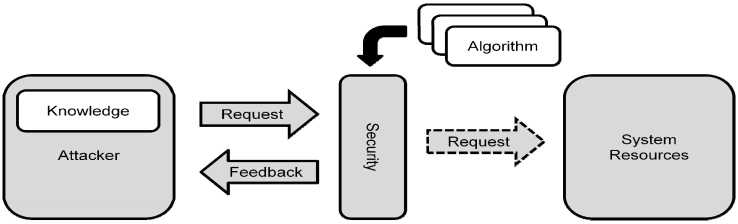

Fig. 1. System Model

System Resources. In the system model, we assume a finite set of resources that can be used, e.g. communication or computation resources. The pool of security algorithms is contained by a security layer deployed to protect the system resources from misuse, e.g. excessively high consumption or consumption for unacceptable purposes. These algorithms classify requests to the system as acceptable or unacceptable based upon a set of rules. If a request is classified as acceptable then the request proceeds and the system resources are consumed. A request that is classified as unacceptable cannot proceed through the security layer, and feedback is provided to the user regarding the failed request.

Attacker. An attacker is an agent (human or computational) that attempts to misuse system resources. The attacker makes requests for the system resources, which pass through the security layer as described above. The attacker has some prior *knowledge* about the rules used to classify request, and attempts to design requests to be classified as acceptable. On each failed attempt, the attacker receives some feedback from the system. This feedback may be a simple Boolean response, or may include reasons for the failure. The attacker can add this feedback to his knowledge, and use this knowledge to inform his subsequent requests. By repeatedly performing this knowledge acquisition process[1], the attacker can derive the rules that are used by algorithms to classify requests. This includes both the parameters used, and the values of these parameters. The attacker can then misuse system resources by sending requests that are structured in such a way that they fulfill the rules of the algorithm in the security layer. In that case, we consider the algorithm ‘broken’.

[1] In the education context, e.g. [23] stated that the learning is driven by a process of inquiry.

Security Layer. To maintain the security of the system, the security layer must be updated. Within this update, the algorithm used by the security layer is replaced by another algorithm from the pool to encapsulate a different set of rules such that requests that are misusing system resources are no longer permitted to pass through the security layer. The attacker must repeat the process of knowledge acquisition in order to determine the new classification rules, such that he can continue sending requests to misuse system resources. This process of learning takes time and the overall aim of the algorithms is to maximize the time until all are broken.

Algorithms. The selection of algorithm when updating the security layer determines the subsequent security of the system. In particular, a set of algorithms D, $D = \{d_1, d_2, ..., d_n\}$, n > 1, $d_n \in D$, represents the security layer of the system. Each of these algorithms has a specific robustness level $dr_i \in DR$. This robustness includes a number of rules $dr_i = \{r_1, r_2, \ldots, r_n\}$, $n \geq 1$ to protect the system. Based on the rules of an algorithm, a set of algorithm is classified into three types: overlapping rules, not overlapping rules or mixed. In the first type, some of the algorithms are subset from each other $dr_i \subset dr_{i+1}$. The importance of this type can be in *breaking up a secure algorithm into a set of algorithm*, and more details about this would be given latter. In the not overlapping rules type, all the algorithms are independent $dr_i \neq dr_{i+1}$. The importance of this type can be in using *variance algorithms*. The third type is that using mixed overlapping rules and not overlapping rules algorithms $dr_i \subset dr_{i+1}$ and $dr_{i+1} \neq dr_{i+2}$. The importance of this type can be, in addition to the importance of the first and second type, releasing an independent algorithm between dependent algorithms may *impair the attacker's learning process*.

The order in which algorithms are released may thus be important. The longer it takes for the attacker to acquire the necessary knowledge regarding classification, the longer the system is protected from misuse. For instance, in the pool of algorithms that can be selected, there may exist algorithms that have some overlapping or similar rules. The question for the defender then is in what order to release these algorithms so that the time until all algorithms are broken is maximized.

3 Experimental Study

In order to test the hypothesis that the time spent on breaking a set of defensive algorithms depends on the order in which these algorithms are released, we conducted a controlled laboratory experiment in which subjects were asked to break a set of three specifically created spam-filter algorithms. The main hypothesis under test is:

> **H_1 – The time it takes to break a series of algorithms is dependent on the order in which the algorithms are released**

We like to point out that the objective of our experiment is not to say anything definitive about spam filter algorithms. We designed bespoke spam filters that would fit the purposes of our experiment, although we have ventured to create an intuitively appealing experiment that has various elements in common with traditional spam filters. Further, an automated program can be used to break the algorithms. However, a form of understanding of a human learning process can be seen clearly by sending e-mails to evade a spam filter, as automated approaches are abstractions of this human learning process that require encoding by human.

3.1 Experiment Setup

The experiment involves subjects to act as potential attackers carrying out attacks on a test system, within which a number of different security algorithms have been deployed.

Experiment Design. In order to evaluate the time needed to break the algorithms, we decided to use a *between-subjects* design. That is, we have two sets of subjects breaking a series of defensive algorithms where the order is different between the groups. This type of design ensures that the exact same algorithms are used in each experiment condition, and that there is no unnecessary confounding factor biasing the results (at the cost of recruiting relatively many participants). The main independent variable for this experiment is the algorithm order. The time consumed to break each algorithm and the numbers of trials are the dependent variables.

The participants are randomly assigned to one of the following two experimental groups:

- **Group 1:** The order of algorithms for this group was: A1, A2 then A3.
- **Group 2:** The order of algorithms for this group was: A2, A1 then A3.

Specifics about the algorithms will be given in the remainder of this section.

Attackers. A nontrivial problem was to find potential attackers. The aim was attackers that could be considered to be non-specialists. Whilst specialist attackers or security experts could have been recruited, they would give us information mostly about where and how our particular algorithms needed to be improved and less about learning. Forty students were recruited for this experiment (34 male and 6 female, something we did not consider relevant for our experiment). The typical age range of subjects was 24-33 with 4 participants in the group 40+. The subjects of this experiment were 40 master and PhD students from the School of computing science and other schools in Newcastle University. 37 subjects have technical backgrounds (majoring in computer science and engineering), and the remaining 3 subjects non-technical (in linguistics). It is important to note that because our aim was to recruit non-specialists attackers to observe the learning acquisition, the inconsistency-bias produced by attackers' background is reduced. In addition, as stated in [15, 23] the learning process is achieved regardless the backgrounds.

System. A challenge in designing the experiment is to design a system that can be breached by ordinary people in a matter of minutes. We found that spam filters could offer a very good model for our experimental requirements. Although as mentioned we do not claim or attempt to study and derive results for spam filters themselves, we do believe the simple spam filters we consider have enough similarities with reality to act as an example of the class of systems that we introduced in Section 2.

We developed a web-based system on which to perform the experiment. A Web application was developed, which enables each participant to perform a registration process (e.g. choosing a username, password and educational background), sign a consent form, and read a brief introductory page with necessary information (e.g. description of the experiment, experiment factors, the participant goal, applied method on how to defeat a content-based spam-filter). The participant can then begin the experimental process, interacting with the spam-filter algorithms. The main idea is that the participants try to send e-mails that pass through the spam filters—we describe this in more detail in Section 3.2. The system records all attempts and the time taken by each participant to break each algorithm in each session.

Algorithms. The rationale behind the defensive spam-filter algorithms we constructed is as follows. A simple algorithm A1 acts as base algorithm and a more complicated algorithm A2 extends the rules used by A1. In other words, the rules in A1 are a subset of A2, which we believe is the first case to consider when one wants to test the hypothesis. The third algorithm A3 does not share any rule neither with A1 nor with A2, and acts as a simple independent algorithm. Hence, we challenge our hypothesis H_1 by the two following questions:

Q_1 – Do G1 and G2 break A1 and A2 within a similar amount of time?

Q_2 – Do G1 and G2 break A3 within a similar amount of time?[2]

We will answer in Section 4 negatively to Q_1 and positively to Q_2. We now describe the specific algorithms A1, A2 and A3, as well as pseudo code describing their operation. Modern density-based spam filter [9] forms the base to implement the algorithms. Note that some of the symbol names being used in the pseudo code are given in Table 1.

Table 1. Symbols used in pseudo code and their values in the experiments

Symbol	Meaning	Value
D	Spam threshold	100
N	Number of hash values for each email	100
S_1	Similarity threshold "Algorithm 1&3"	75%
S_2	Similarity threshold "Algorithm 2"	45%

Algorithm 1 (A1). This algorithm is a simple implementation of the proposal of [9] where only the first part of the message is checked for similar hashes. Further, the similarity threshold is 75. The pseudo code of this is shown in Figure 2.

Algorithm 2 (A2). This algorithm is similar to A1 except that before any calculation of the hash values, the message would go through word transformation that

[2] We have an additional reason to consider this particular question, namely whether we can justify the use of a Markov decision model with a state space that simply keeps track of which algorithms are broken. The question corresponds to demonstrating the memoryless property of the Markov model with that state space. The Markov decision model itself is outside the scope of this paper, we refer to our work in [10] for more details.

would delete all redundant letters, white spaces, unify letters case, and transform common number shortcuts to their equivalent letters (e.g. 4 would become for). Those transformations would create a harder algorithm since it would detect any attempt of the attacker to trick the spam filter by using those word transformations. Furthermore, the similarity threshold is 45. In other words, this algorithm has more rules to increase the robustness level. The pseudo code of this is shown in Figure 3.

```
Input:  T: Text of Mail
           Var h: Hash value
Output: R: result of detection
New-Hash-DB-Candidate ← Make N Hash values from T
For h in New-Hash-DB-Candidate do
    For each first 25 hash in New-Hash-DB-Candidate do
      If $h_i$ in H1 is similar to $h_j$ in H2
      Then increment similarity, increment j and i=j
          Else increment j
          If H1 and H2 share $S_1$ same hash value
          Then R= detected; Update-Similar-Mail (Mail in Hash-DB pointed by h)
          If No. of Similar Mail > D
          Then Mark Hash-DB as "spam"
          Else R= no similarity // If No Similar Entry exists in Hash DB
               //Store-New-Mail (New-Hash-DB-Candidate)
Return R;
```

Fig. 2. Pseudo code of A1

Algorithm A1 and A2 are the main two algorithms to test our hypothesis. We created a third algorithm A3 that both groups G1 and G2 are tasked to break after breaking A1 and A2. Any difference between the groups in the time to break A3 would be very interesting, because it would suggest that the order in which A1 and A2 were broken in the past determines the success of breaking a future algorithm.

```
Input:  T: Text of Mail
            Var h: Hash value
 Output: R: result of detection
 // Remove all white spaces; make the whole text  lowercase
 T' = Normalise (T)
 //Remove triple letters; convert some numbers to letters  (like 4 to for)
 New-Hash-DB-Candidate ← Make N Hash values from T
 For h in New-Hash-DB-Candidate do
      For each first 25 hash in New-Hash-DB-Candidate do
            If $h_i$ in H1 is similar to $h_j$ in H2
            Then increment similarity, increment j and i=j
            Else increment j
            If H1 and H2 share $S_2$ same hash value
            Then R= detected;
            Update-Similar-Mail (Mail in Hash-DB pointed by h)
            If No. of Similar Mail > D
            Then Mark Hash-DB as "spam"
            Else R= no similarity// If No Similar Entry exists in Hash DB
                 //Store-New-Mail (New-Hash-DB-Candidate)
 Return R;
```

Fig. 3. Pseudo code of A2

Algorithm 3 (A3). This algorithm is a simple implementation of the proposal of [9]; however the last part of the message is checked for similar hashes, and the similarity threshold is 75. The pseudo code of this is shown in Figure 4.

It is worthwhile to note that the reason behind using a different threshold is that we empirically found that the similarity threshold can play an important role in terms of determining the difficulty of an algorithm. That is, in A1 and A3, the attacker needs to modify only 25% from the part that the algorithm is checking, while it is 55% in A2.

Briefly, the symbols in Table 1 are explained in the following. As in [9], a hash-based vector representation was used. That is, for each e-mail, hash values of each length 9 substring are calculated[3], and then the first N of them are used as vector representation of the e-mail. To check a single e-mail, in order to find similar previous e-mail which share S% of the same hash values, the algorithm checks the database. As a result, an e-mail transferred more than D times is marked as spam.

```
Input:  T: Text of Mail
           Var h: Hash value
Output: R: result of detection
For read T until the last part
New-Hash-DB-Candidate ←Make N Hash values from the last part of T
For h in New-Hash-DB-Candidate do
    For each first 25 hash in New-Hash-DB-Candidate do
      If $h_i$ in H1 is similar to $h_j$ in H2
      Then increment similarity, increment j and i=j
          Else increment j
          If H1 and H2 share $S_1$ same hash value
          Then R= detected;
            Update-Similar-Mail (Mail in Hash-DB pointed by h)
         If No. of Similar Mail > D
         Then Mark Hash-DB as "spam"
            Else R= no similarity
      // If No Similar Entry exists in Hash DB
      Store-New-Mail (New-Hash-DB-Candidate)
Return R;
```

Fig. 4. Pseudo code of A3

Materials: Stimulus and Rational. The stimulus material provided to participants consisted of some default e-mail text. The subjects were asked to send this text to the server, as if it was a typical e-mail. The e-mail text was chosen to be 512 characters in length. Although real-life spammers may send messages that are shorter than this, the length of messages provides the subjects with sufficient text to utilize a range of different strategies to breach the spam-filter.

The same e-mail text was assigned to all subjects, rather than allowing each subject to write his own e-mail. There were several reasons for this. First, self-written e-mails may be of different lengths, making the measurement and comparison of participant's learning a difficult task. Second, self-written e-mails might be chosen because they are easy to type (or, in perverse cases, particularly hard to type). This would again introduce biases that are difficult to control. Third, the use of the same e-mail template across all subjects means that each subject can be treated as an impostor for all the other subjects, putting testing on a firm foundation. Finally, using the same e-mail for everyone affected experimental control over unanticipated biases.

3.2 Experimental Procedure

In this section, the way we ran the experiment is explained, i.e., instructions to subjects, procedures and the data collected.

[3] We use the standard hash function provided in Java library.

Instructions to Subjects. Subjects were instructed to act as attackers whose target is to defeat the spam-filter algorithms by successfully passing the spam filter algorithms for 3 e-mails (where each e-mail is interpreted as a batch of 100). The subjects were instructed that to defeat an algorithm, they should introduce enough changes to the provided message template to trick the spam filter into thinking that the message being sent is genuine. The maximum number of changes they were allowed to introduce at each trial was 80. This makes it impossible for participants to write a completely different message. Moreover, the copy and paste functions were not activated to avoid sending completely different e-mail.

Subjects were instructed that there are a number of candidate attacks that spammers can enact to fool spam filter algorithms. For example [24]: *Random addition*, *Thesaurus substitution*, *Perceptive substitution* and *Aimed addition*. In this way we aimed at creating a level playing field for all participants.

Subjects were told that if they needed a break; they were to do so *after* they had defeated all the algorithms. Subjects were able to gauge their progress by looking at a counter at the right of the screen which showed how many e-mails had been sent successfully so far and how many yet remained. Subjects were admonished to focus on the task and to avoid distractions, such as talking with the experimenter, while the task was in progress.

Procedures. The experiment was conducted in a controlled laboratory environment to avoid any distractions, and collect the desired data without any biases. As we mentioned, each group had an equal number of participants (i.e. 20 participants). Each participant was offered £5 for the participation. To motivate participants to do their best, like real attackers, an additional incentive to increase their motivation was offered. The participant who got the highest score in each group was awarded £40 while the second ranked subject was awarded £20. The highest score is based on the time and number of trials consumed to complete the task.

After every e-mail message send by the participant, the system gives information about the progress made: whether the spam attempt passes or fails, and once the algorithms is considered defeated, a notice that the deployed algorithm of the system has been changed. At the end of the experiment, each participant was informed about the achieved score, the time taken and the number of trials. Finally, the participant was asked to fill a short survey/questionnaire about his or her experience.

Collected Data. The time taken by each participant to defeat the algorithms in each session was recorded by the system. Further, the number of trials and the e-mails sent for each session were recorded as well (for later analysis). Additionally, the questionnaires were collected.

4 Results and Analysis: Does Order Matter?

In the experimental study, all the participants successfully completed their task. We first discuss in Section 4.1 the hypothesis with respect to the ordering of the algorithms A1 and A2. We then discuss in Section 4.2 the hypothesis regarding the insensitivity of the order of A1 and A2 with respect to the time used to defeat A3. Further, the impact of order on defeating all algorithms is presented in Section 4.3.

4.1 Testing Hypothesis If Order Matters

The average time needed to break each algorithm in the two groups is shown in Figure 5. From the totals (the rightmost bar), we see that Group 1 took longer than Group 2. This indicates there are implications to the ordering of the algorithm, but we will discuss this more precisely below. As expected, the 'tougher' algorithm A2 took more time to break than A1. In Group 2 it took on average 16.2 minutes and in Group 1, 14.1 minutes. The time needed to break A1 is far less in Group 2, possibly because learning of the techniques to break A2 first, which is effectively a superset of A1, is enough to break A1. The statistical significance of this will be discussed now.

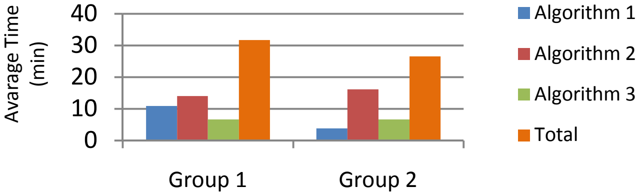

Fig. 5. The average time (in minutes) for 'attackers' in each group to break the algorithms

Table 2. Order matters of two algorithms A1 and A2

Group	Total time				Total trials			
	Avg.	SD	Max	Min	Avg.	SD	Max	Min
1	25.0	10.6	57.1	13.5	33.4	20.1	99.0	14.0
2	20.1	4.3	26.0	10.8	26.1	8.3	40.0	11.0

Table 2 compares the algorithms A1 and A2 in the two groups, with respect to time needed (middle column) as well as trials made (rightmost column). For both, we provide average (Avg.), standard decision (SD) and maximum (Max) and minimum (Min). With respect to average time, we find the following. Time needed for breaking A1 and A2 in Group 1 is 25.0 minutes, while it is 20.1 minutes in Group 2. A t-test yields a result of t=1.89, p<0.1, indicating that the difference between Group 1 and Group 2 is indeed statistically significant.

We can therefore answer negatively to Q_1 by observing that G2 break A1+A2 with significantly less time than G1. This result validates the hypothesis H_1, since we exhibit a case where the order of release has an impact on the time required to break an algorithm.

With respect to the number of trials, we found a less significant difference. The average number of trials is 33.4 trials for Group 1 and 26.1 trials for Group 2. However, this difference is not statistically significant (t=1.55, p=0.139). The discrepancy between time and trials is interesting; we do not believe that this invalidates our claim that order matters but it shows that it is not always apparent (and, of course, as always, possibly not true). We discuss this a bit more at the end of this section.

Table 3. Breaking A1 for each group

Group	Total time A1				Total trials A1			
	Avg.	SD	Max	Min	Avg.	SD	Max	Min
1	10.9	4.7	24.8	5.5	14.8	6.8	31	6
2	3.8	1.3	6.5	0.86	4.3	1.87	8	2

If we then look at A1 and A2 individually, we find that the difference in the total time/trials can be attributed particularly to the time/trials it takes to break A1. Comparing the time and trials to break A1 in the two groups in Table 3, a t-test yields a result of t=6.33, p<0.001, indicating that the time consumed to break A1 in Group 1 is significantly higher than that in Group 2. Also a statistically significant difference is found in the number of trials (t=6.62, p<0.005).

Table 4. Breaking A2 for each group

Group	Total time A2				Total trials A2			
	Avg.	SD	Max	Min	Avg.	SD	Max	Min
1	14.1	6.2	32.3	7.8	18.6	14.7	74	8
2	16.2	3.4	21.3	8.9	21.7	7.2	34	9

Likewise, we compare A2 in the two groups in Table 4. A t-test yields a result of t=-1.32, p=0.196, indicating that the time consumed to break A2 in Group 2 is not significantly higher than that in Group 1. The difference found in number of trials was not found a statistically significant (t=-0.86, p=0.399). In other words, the difference for A2 is less significant than that for A1. This suggests that attackers learn more from A2 than from A1, in terms of how much it speeds them up in attacking the other algorithm.

It is worthwhile to note that previous research assumed, based on psychology studies, that interacting with a limited set of highly similar exemplars leads to more learning than when the instances are distributed and dissimilar [16]. Our order matters results appeared to confirm this assumption. The average time of breaking A1 in Group 2 was 3.8 minutes, and the maximum time was 6.5 compared to 10.9 minutes and the maximum time was 24.8 minutes in Group 1. On the contrary, the average time of breaking A2 in Group 2 was 16.2 minutes instead of 14.1 minutes in Group 1.

4.2 The Influence of Order on Defeating Future Algorithms

The negative answer to Q_1 validates the hypothesis H_1; however we also need to verify that the difference between in G1 and G2 comes indeed from the release order, and not from the fact that G2 contains more naturally talented attackers. To research this question, we added the third algorithm A3 at the end of each experiment. This allows us to check if the order of the previous algorithms has any effect on the time needed to defeat the subsequent algorithm A3.

Table 5. Breaking A3 for each group

Group	Total time A3				Total trials A3			
	Avg.	SD	Max	Min	Avg.	SD	Max	Min
1	6.70	4.93	17	0.97	8.8	10.3	49	2
2	6.5	4.4	16.3	0.56	6.05	4.4	20	2

We compare A3 in the two groups in Table 5. A t-test yields a result of t=0.14, p=0.891, indicating that there is no statistically significant difference between the times needed to break A3 in Group 1 and Group 2, respectively. Also, no statistically significant difference was found in the number of trials (t=1.12, p=0.273). Hence, we can positively answer to Q_2, by observing that G1 and G2 take a similar amount of time to break the independent algorithm A3.

One needs to be careful to generalize this result: we do not claim here that any independent algorithm would require the same amount of time, regardless of the history of the attackers. It may be that if A3 was more closely related to A1 and A2, the results will be different. In addition, one would expect that aspects that influence the working of memory of the attacker may matter, such as the absolute time it takes to break algorithms. After all, it is not unlikely that attackers simply forget more of the knowledge gained from earlier attacks if the attack is longer in the past. So, we look at this experiment as an initial look at this issue.

4.3 The Influence of Order on Defeating All Algorithms

To complete our discussion, we revisit the influence of ordering for the time and effort it takes to break all three algorithms.

Table 6. Breaking all algorithms for each group

Group	Total time				Total trials			
	Avg.	SD	Max	Min	Avg.	SD	Max	Min
1	31.7	10.8	59.4	17.4	42.3	23	112	16
2	26.6	7.36	36.1	12.2	32.1	11	56	13

We compare the effects of all algorithms in the two groups in Table 6. A t-test yields a result of t=1.76, p<0.1, indicating that the time needed to break the series of algorithms in Group 1 is significantly higher than in Group 2. The average number of trials is 42.3 trials in Group 1 compared to 32.1 trials in Group 2 and a t-test yields a result of t=1.78, p<0.1, indicating that the number of trials in Group 1 is statistically significantly higher than in Group 2. This is somewhat surprising, since the time to break A3 differs little between groups, and we showed in Section 4.1 that without algorithm A3 the difference in the total numbers of trials of the two groups is not statistically different. This may suggest that with respect to the number of trials needed the validity of our hypothesis is at the edge of statistical significance.

5 Discussion

Our experimental study provides statistically significant evidence that the release order for a set of algorithms can increase the time needed to break a system's security. In particular, as shown in Table 2, the time required by Group 1 to break the algorithms was significantly higher than Group 2. So, we established the main objective of this paper, namely that 'order matters'.

Since A1 is a simplified version of A2, our experiments also indicate that the success of attacks can be delayed by breaking up an algorithm in parts that are released in sequence. We have to be careful not to generalize that conclusion too quickly, but it is an interesting insight that would imply the intuitive reasoning that by breaking up an algorithm in subsets you 'teach' the attacker how to attack is less valid.

It is important to point out that the value in testing the two conditions (i.e. A1 is deployed before A2 and A2 is deployed before A1) in which the defenses are overlapping was necessary to build our hypothesis on solid grounds before we conduct further experiments. Even though several studies in the psychology and education filed indicated that the learning curves can be considerably increased by interacting with a limited set of highly similar exemplars [e.g. 15, 16], these studies were only focused on the effectiveness of the presentation order on the categorization models. For this reason, in our experiment, even the trivial assumption (i.e. A2 is deployed before A1) is tested to avoid surprises. Hence, the experimental results achieved from these conditions can lead to a further experiment in which the algorithms order is, for instance, subset, independent and superset or other orders, as will be highlighted in the following.

Furthermore, based on the qualitative data, we found that the participants performed the attacking process by strategies using skills gained. Among such wrongful direction is the believing that the algorithm is checking a different part of the e-mail. Interestingly, pervious research assumed, based on empirical results, that the attackers' skills would increase based on the knowledge acquired [5]. Our qualitative data appeared to confirm this assumption.

The concatenation of A3 at the end of both Group 1 and Group 2 yielded an interesting and important result. It showed that despite the knowledge gain at any point of the release chain, injecting a non-subset algorithm would force the attacker back to the learning phase. Accordingly, we investigated in [21] such orders, for examples, A1, A3 and A2 or A2, A3 and A1 that lead to more interesting results. Further, we also note that we used the insight that breaking A3 takes an equal amount of time for both group as a confirmation that a Markov model is an appropriate formalism for the problem at hand, as we shown in [10].

As the results obtained in this paper are very encouraging, optimizing the release order of defensive algorithms is a problem worthwhile to study. The results in this paper are a first step, showing the validity of the problem and providing insights in where to invest future research.

6 Related Work

Attack Modeling. A quantitative analysis of attacker behavior based on empirical data collected from intrusion experiments was presented in [2]. Beside this, Ortalo et al.

described in [6] a technique where the states of the resulting Markov chain denote the enhanced privileges gained by an attacker as a result of series of atomic attacks on a system. Furthermore, generic models have been developed that focus on evaluating security [3, 1]. In relation to the time taken for an attacker to compromise a system and misuse its resources, several studies proposed in [4, 5, 14] a model for estimating the time to compromise a system component that is visible to an attacker. Additionally, McQueen et al. suggested in [4, 5] that the attacker skill levels should be consideration when determining the mean time to compromise a system. With regard to adversarial machine learning, a recent study by Huang et al. explored in [19] that the limits of an adversary's knowledge about the machine learning algorithm and the input data.

Game Theoretic Security Approaches. In traditional network security solutions, one of the first approaches for applying game theory to network security is discussed in [11]. Furthermore, Alpcan and Baser utilized in [12] Min-max Q learning to aid in the gradual improvement of the defender's quality. This work can handle reactive defense actions, while in [13] proposes proactive defense measures. In addition, a study by Jiang et al. [18] focused on active defense using an approach to attack prediction. Furthermore, a game theory approach has been adapted to attack modeling as a means for computing and therefore predicting the expected attacker strategy [8]. Recently, a comprehensive survey has been conducted by Roy et al. [25] concluded that most of the current games theoretic are based on static game, games with perfect information or games with complete information. Indeed, although a number of current models involving dynamic games with incomplete and imperfect information exist, none of them considers learning and/or maximizing the duration of the game as the game's objective.

Effects of Information Order. The idea that the order in which information is received could affect both the learning process and the ultimate knowledge representation is of course commonly studied in the fields such as education or psychology. Most of this literature focused on the similarity-based order. The similarity-based order that maximizes the adjacency of the training examples has investigated by numerous studies [15, 16, 17]. An empirical non-linguistic experiment by Elio and Anderson evaluated in [15] the effects of information order and variance on schema abstraction. This research shown that the low-variance condition group performed better with regards to typicality ratings and accuracy than high-variance condition. Further, this study concluded that the manipulation of presentation orders is a potentially useful tool for studying the mechanisms of learning. Mathy and Feldman recently showed in [20] that a rule-based presentation order (i.e. a small number of rule-learning assumptions) yields superior learning compared to the similarity-based order. Despite these studies provided some insights into presentation orders for investigating categorization processes concept, our study is clearly investigated the effects of presentation order on different concept type.

7 Conclusion and Future Work

This paper reports on experimental validation of the hypothesis that the order in which defensive algorithms are released impacts the success of the attacker. Our work is based on the observation that attackers increase knowledge by learning from their attempted attacks, and on the intuition that the learning experience of attackers can be influenced by the order in which defensive algorithms are released.

Through a between-subjects experiment with simplified but representative spam filter algorithms we were able to show that the order in which defensive algorithms are released indeed influences the time attacks take. This is a very encouraging result for this line of research, indicating that the problem merits study. The experiment also provides an indication that breaking up a defensive algorithm can be a beneficial tool in prolonging the overall attack time, but this issue need to be researched in much more detail before this conclusion can be drawn more widely. Furthermore, the experiment shows that the success in breaking future algorithms does not depend on how that total amount of knowledge was gained.

A number of potential issues for future research follow from our research. Some of the challenges are of technical nature, but the largest challenge may lie in gaining a deeper understanding of the way attackers collect knowledge (i.e., in the way they learn). That would allow us to better estimate the time it takes to break a defensive algorithm under various levels of knowledge gained, and would allow us to determine an optimal strategy without conducting the time-consuming experiments carried out in this paper. Similarly, it will be of interest to investigate deeper if breaking up defensive algorithms in 'subsets' indeed increases the speed at which attackers gain knowledge, as we found in the our experiments.

References

1. Almasizadeh, J., Azgomi, M.A.: Intrusion Process Modeling for Security Quantification. In: 4th the International Conference on Availability, Reliability and Security, pp. 114–121. IEEE Press, Los Alamitos (2009)
2. Jonsson, E., Olovsson, T.: A Quantitative Model of the Security Intrusion Process Based on Attacker Behavior. IEEE Transactions on Software Engineering 23, 235–245 (1997)
3. Madan, B.B., Goseva-Popstojanova, K., Vaidyanathan, K., Trivedi, K.S.: Modeling and Quantification of Security Attributes of Software Systems. In: IEEE International Conference on Dependable Systems and Networks, pp. 505–514. IEEE Press, Los Alamitos (2002)
4. McQueen, M.A., Boyer, W.F., Flynn, M.A., Beitel, G.A.: Time-to-Compromise Model for Cyber Risk Reduction Estimation. In: First Workshop on Quality of Protection, pp. 49–64. Springer, Milan (2005)
5. McQueen, M.A., Boyer, W.F., Flynn, M.A., Beitel, G.: Quantitative Cyber Risk Reduction Estimation Methodology for a Small SCADA Control System. In: 39th Annual Hawaii Conference on System Science, IEEE Press (2006)
6. Ortalo, R., Deswarte, Y., Kaaniche, M.: Experimenting with Quantitative Evaluation Tools for Monitoring Operational Security. IEEE Transactions on Software Engineering 25, 633–650 (1999)
7. Alsuhibany, S.A.: Optimising CAPTCHA Generation. In: 6th International Conference on Availability, Reliability and Security, pp. 740–745. IEEE Press (2011)

8. Sallhammar, K., Knapskog, S.J., Helvik, B.E.: Using Stochastic Game Theory to Compute the Expected Behavior of Attackers. In: International Symposium on Applications and the Internet Workshops (2005)
9. Yoshida, K., Adachi, F., Washio, T., Motoda, H., Homma, T., Nakashima, A., Fujikawa, H., Yamazaki, K.: Density Based Spam Detector. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 486–493. ACM (2004)
10. Alsuhibany, S.A., Alonizi, A., Morisset, C., van Moorsel, A.: Optimizing the Release Order of Defensive Mechanisms. In: 29th Annual UK Performance Engineering Workshop (to appear)
11. McInerney, J., Tubberud, S., Anwar, S., Hamilton, S.: FRIARS: a Feedback Control System for Information Assurance Using a Markov Decision Process. In: 35th Annual 2001 International Carnahan Conference on Security Technology, pp. 223–228. IEEE Press (2001)
12. Alpcan, T., Baser, T.: An Intrusion Detection Game with Limited Observations. In: 12th International Symposium on Dynamic Games and Applications, France, (2006)
13. Shiva, S., Roy, S., Dasgupta, D.: Game Theory for Cyber Security. In: Sixth Annual Workshop on Cyber Security and Information Intelligence Research (2010)
14. Leversage, D.J., Byres, E.J.: Comparing Electronic Battlefields: Using Mean Time-to-Compromise as a Comparative Security Metric. In: Gorodetsky, V., Kotenko, I., Skormin, V.A. (eds.) MMM-ACNS 2007. CCIS, vol. 1, pp. 213–227. Springer, Heidelberg (2007)
15. Elio, R., Anderson, J.R.: The Effects of Information Order and Learning Mode on Schema Abstraction. Memory and Cognition 12, 20–30 (1984)
16. Sandhofer, C.M., Doumas, L.A.A.: Order of Presentation Effects in Learning Color Categories. Journal of Cognition and Development 9, 194–221 (2008)
17. Medin, D.L., Bettger, J.G.: Presentation Order and Recognition of Categorically Related Examples. Psychonomic Bulletin & Review 1, 250–254 (1994)
18. Jiang, W., Tian, Z., Zhang, H., Song, X.: A Stochastic Game Theoretic Approach to Attack Prediction and Optimal Active Defense Strategy Decision. In: IEEE International Conference on Networking, Sensing and Control, pp. 6–8. IEEE Press (2008)
19. Huang, L., Joseph, A.D., Nelson, B., Rubinstein, B.I.P., Tygar, J.D.: Adversarial Machine Learning. In: 4th ACM workshop on Artificial Intelligence and Security, pp. 43–58 (2011)
20. Mathy, F., Feldman, J.: A Rule-Based Presentation Order Facilitates Category Learning. Psychonomic Bulletin & Review 16, 1050–1057 (2009)
21. Alsuhibany, S.A., van Moorsel, A.: Modelling and Analysis of Release Order of Security Algorithms Using Stochastic Petri Nets. In: 8th International Conference on Availability, Reliability and Security (to appear)
22. Caliendo, M., Clement, M., Papies, D., Scheel-Kopeinig, S.: The Cost Impact of Spam Filters: Measuring the Effect of Information System Technologies in Organizations. Information Systems Research, 1–13 (2012)
23. Kahn, P., O'Rourke, K.: Guide to Curriculum Design: Enquiry-Based Learning. Higher Education Academy, York (2004)
24. Garcia, F.D., Hoepman, J.-H., van Nieuwenhuizen, J.: Spam Filter Analysis. In: 19th IFIP International Information Security Conference, Toulouse, pp. 395–410. Springer (2004)
25. Roy, S., Ellis, C., Shiva, S., Dasgupta, D., Shandilya, V., Wu, Q.: A Survey of Game Theory as Applied to Network Security. In: Hawaii International Conference on System Sciences, pp. 1–10. IEEE Press (2010)