Optimising the Release Order of Defensive Mechanisms

Suliman A. Alsuhibany, Ahmad Alonaizi, Chris Smith and Aad van Moorsel
Optimising the Release Order of Defensive Mechanisms

S.A. Alsuhibany, A. Alonaizi, C. Smith and A. van Moorsel

Abstract

In the practical use of security mechanisms such as CAPTCHAs and spam filters, attackers and defenders exchange 'blows', each celebrating (temporary) success in breaking and defending. We are interested in the question of whether the order in which defensive algorithms are 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 experience. 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. We then model the problem as an optimization problem using a Markov Decision Process model. We present a tailored optimization algorithm to obtain efficiently the optimal release strategies for any given model.
Bibliographical details

ALSUHIBANY, S.A., ALONAIZI, A., SMITH, C., VAN MOORSEL, A.

Optimising the Release Order of Defensive Mechanisms
[By] S.A. Alsuhibany, A. Alonaizi, C. Smith, A. van Moorsel

(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1333)

Added entries

NEWCASTLE UNIVERSITY

Abstract

In the practical use of security mechanisms such as CAPTCHAs and spam filters, attackers and defenders exchange ‘blows’, each celebrating (temporary) success in breaking and defending. We are interested in the question of whether the order in which defensive algorithms are 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 experience. 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. We then model the problem as an optimization problem using a Markov Decision Process model. We present a tailored optimization algorithm to obtain efficiently the optimal release strategies for any given model.

About the authors

Suliman A. Alsuhibany is a PhD student in the school of computing science under the supervision of Prof. Aad van Moorsel in the area of Security, primarily about developing an approach for defensive mechanisms that are expected to be broken over time. He received his BSc in Computer Science from Al-Qassim University, Saudi Arabia. In 2009 he received his MSc in Computer Security and Resilience from Newcastle University. Suliman is interested in game theory, attack modelling and empirical approaches for quantitative evaluation security systems.

Ahmad Alonaizi is a PhD student at the Cybercrime research lab in Newcastle University. He joined Newcastle University after receiving his MSc degree in Web Technologies from the University of Southampton in 2010. Before that, he spent about a year as an IT project Manager in Afghanim Industry, A mega family conglomerate that is based in Kuwait. His Bsc degree was in Computer Engineering - Software Option which he obtained from Concordia University - Canada in 2008. Currently he is sponsored by the Public Authority for Applied Education and Technology, PAAET, in Kuwait where he works as a tutor.His current research interests include: Trust Management Frameworks, Web Development, and Online Privacy.

Chris Smith was a Research Associate in the School of Computing Science, Newcastle University.

Aad van Moorsel is a Professor in Distributed Systems and Head of School at the School of Computing Science, which he joined in 2004 having previously been a postdoc at University of Illinois in Urbana-Champaign, researcher at Bell Labs/Lucent Technologies in Murray Hill and then a research manager at Hewlett-Packard Labs in Palo Alto, all in the United States. His research group develops tools and techniques to make Internet use safer for people and businesses, mostly based on quantitative methods. He is the Newcastle lead of the GCHQ/EPSRC Research Institute in the Science of Cyber Security, PI for the EPSRC Network on Cyber Crime, Technology Director for the Social Inclusion through the Digital Economy hub and PI for the EPSRC sponsored Hyper-Privacy project with a local women's charity.

Suggested keywords

SECURITY AND PROTECTION
ALGORITHMS
LEARNING
EXPERIMENTATION
OPTIMISATION
ABSTRACT
In the practical use of security mechanisms such as CAPTCHAs and spam filters, attackers and defenders exchange "blows," each celebrating (temporary) success in breaking and defending. We are interested in the question of whether the order in which defensive algorithms are 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 experience. 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. We then model the problem as an optimization problem using a Markov Decision Process model. We present a tailored optimization algorithm to obtain efficiently the optimal release strategies for any given model.

Categories and Subject Descriptors
D.4.6 [Operating Systems]: Security and Protection — access controls, authentication;

General Terms
Algorithms, Design, Reliability, Experimentation, Security

Keywords
Spam, CAPTCHA, Learning, Optimization

1. INTRODUCTION
The primary purpose of the security algorithms that are considered in this paper is to protect system resources from misuse. The algorithms encode a set of rules that characterize this 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.

As observed in [2], when a new attack is successful, knowledge of this attack is distributed to potential attackers. This algorithm will then be useless in protecting the system from misuse. A replacement defensive algorithm is needed. This raises the following question: can we help the developer and planner of the defensive algorithms? 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?

The main hypothesis behind our work is that the time taken to break a set of defensive algorithms depends on what the attacker has learned from earlier successful attacks on algorithms. Moreover, we hypothesize that we can influence the learning done by the attacker by the way we order the release of the defensive algorithms. In this paper we formulate and solve the optimization problem based on both of these issues.

To research this problem we first test that the hypothesis is correct, or at least demonstrate it is believable. For that reason, we conduct an experiment with simplified but representative spam filter algorithms. We ask two groups of twenty subjects to break these algorithms, where the algorithms are presented in a different order to the respective groups. The experiment shows that there is a statistically significant difference in the time taken by the two groups to break the algorithms. So, it confirms the hypothesis that the total time for an attack to succeed depends on the order in which defensive algorithms are released (or, to be more precise, the hypothesis is not rejected [16]).

Our second contribution is to provide an optimal strategy for the release of defensive algorithms. Our approach to this task is to mathematically model the optimization problem, and to present a bespoke efficient solution algorithm that derives the optimal release strategy for any model. Our model is a Markov Decision Process model, with a specific state space that we utilize to derive the efficient optimization algorithm.

To the best of our knowledge, we are the first to address this particular issue of optimizing the release strategy to delay the attack success as long as possible. There exists a considerable amount of related work that considers the attack and defense interaction as a game-theoretic problem, but these formulations do not fit exactly with our approach. In particular, the existing literature does not explicitly represent learning in the model. It also typically does not consider maximizing the duration of the game as the objective, but aim for Nash or other equilibria. This does not fit our perspective of finite sets of algorithms that need scheduling. (We refer to Section 7 for a further discussion of related work.)

The rest of this paper is organized as follows. Section 2 describes the system model that forms the basis of our experimental study and our derivation of optimal release strategies. Section 3 describes the experimental study, and its results are presented in Section 4. Our approach to the derivation of optimal release strategies is presented in Section 5. The discussion is presented in Section 6. Section 7 discusses related work. Finally, we conclude with overall discussions and future work in Section 8.
2. SYSTEM MODEL
In order to describe our approach, we start by providing an abstract system model of the attack scenario, involving the attacker and the system as shown in Figure 1. This model provides the basis of the experimental study that we present in Section 3, and is refined into a stochastic model in Section 5 to derive optimal release strategies.

![Figure 1. System Model](image)

The aim of the system model in Figure 1 is to describe 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. In the system model we assume a finite set of resources that can be used, e.g. communication or computation resources. A security layer is deployed to protect the system resources from misuse, e.g. high consumption or consumption for unacceptable purposes. The security layer contains a pool of algorithms. These algorithms classify requests to the system as acceptable or unacceptable based upon a set of rules. A request that is classified as acceptable can proceed through the security layer and use system resources. 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, and feedback is provided to the user regarding the failed request.

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 structure his 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, 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.

Therefore, to maintain the security of the system, the security layer must be periodically 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.

The order in which algorithms are released may thus be important. The more time 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: SPAM-FILTER ALGORITHMS
In order to test the hypothesis that the time spent on breaking some defensive algorithms depends on what the attacker has learned from earlier successful attack algorithms, and that we can influence the learning done by the attacker by the way we order the release of the defensive mechanisms, we conducted a controlled laboratory experiment in which subjects were asked to break a set of three specifically created spam-filter algorithms.

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. Specifically, the attackers are asked to formulate and dispatch spam messages through a mail server, on which different spam-filter algorithms were deployed. When the attackers collect sufficient knowledge to adapt their interactions and misuse the system and dispatch their spam messages, the spam-filter algorithms are replaced and the attackers are told to continue their attacks. The time taken for the attackers until they are able to misuse the system under different spam-filter algorithms was collected. The algorithms were presented to two different groups of subjects in different order, and we recorded the time and attempts taken to break the algorithms. Based on these data points we determine whether there is a significant difference in the time taken between the groups. Moreover, this enables us to determine whether or not the time taken to misuse the system is dependent on the order in which the algorithms are deployed.

3.1.1 Experiment Design
In order to evaluate the time needed to break the algorithms, we decided to use a between-subjects design. This type of design requires more participants but ensures that the exact same algorithms are used in each experiment condition, and that there is no unnecessary confounding factor biasing the results. The main independent variable for this experiment is the algorithm order. The time consumed to break each algorithm and the numbers of trails are the dependent variables.

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

- **Ascending Group (G1):** The order of algorithms for this group was: A1, A2 then A3.
- **Descending Group (G2):** The order of algorithms for this group was A2, A1 then A3.

Given these two experimental groups, our hypothesis was:
H1 – The time taken to break a series of algorithms is dependent on the order in which the algorithms are released.

We explain the rationale behind the three algorithms and their ordering in the respective groups in more detail in Section 3.1.4.

3.1.2 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). A brief instructions page was shown at the beginning of the experiment to explain the basics of how to break a spam filter. The participant could view the same instructions again at any moment during the experiment. This step was necessary to minimize the results bias by insuring that all participants are starting from a common knowledge background.

3.1.3 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 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 we introduced in Section 2.

Modern spam filters [17] work by recording hashes from each message being sent through the mail server. If a new message has a certain percentage of hashes similar to a previously recorded message, the new message would be considered as similar to the previous one. If then a similar message have been found more than a certain number of times, it is considered as spam. The server would then delete any new message that is sufficiently similar.

To try to break spam filters, one can use simple writing techniques, such as random letter additions, thesaurus substitutions, or whitespace injection. When applied to a message, these techniques result in different hashes and, hence, trick the spam filter into thinking that this message is not similar to an old (spam) message. Our experiment evaluates how quick subjects find out how to edit messages so that it defeats the spam filter algorithm.

We developed a web-based system on which to perform the experiment. A Web application called SpamDefender 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 that includes an explanation about the 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 could then begin the experimental process, interacting with the spam-filter algorithms.

The proposed algorithm in [17] was chosen as the base to implement three different, but related, algorithms that are based on the core idea of comparing hashes similarities. We note that this algorithm has been demonstrated to have a 98% recall rate and 100% precision by using an unsupervised learning engine.

3.1.4 Algorithms
The rationale behind our spam filter algorithms 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. Intuitively, if we release algorithm A1 before A2, one could argue that the attackers learn from A1 to break A2, and therefore break the two algorithms together quicker. One could also argue that the learning from A1 may distract from breaking A2, for instance if the attacker does not realize the similarity or (conversely) becomes preconditioned to only considered particular attack patterns. The experiment will show us that in this particular case, the latter reasoning seems most valid.

We now describe the specific algorithms A1 and A2, as well as pseudocode describing their operation. Note that some of the variable names being used in the pseudo code are given in Table 1.

Algorithm 1 (A1): This algorithm is a simple implementation of the proposal of [17] where only the first part of the message is checked for similar hashes. 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 would delete all redundant letters, white spaces, unify letters case, and transform common number shortcuts to their equivalent letters (e.g. 4 would become 4). 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. The pseudo code of this is shown in Figure 3.

```plaintext
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 in H1 and H2 share S; same hash value
      Then R= detected;
      Update-Similar-Mail (Mail in Hash-DB pointed by h)
    Else R= no similarity
      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;
```

Figure 2. Pseudocode of A1.
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, in H1 is similar to h in H2
      Then increment similarity, increment j and i=j
    Else increment j
    If H1 and H2 share S1 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;

Figure 3. Pseudocode of A2.

Without going into too much detail, one can see from Figure 2 and Figure 3 that the algorithms have significant similarity, with A1 using a subset of the rules of A2. As we explained previously, the only differences are processing A2 through a word transformation, and decreasing the similarity threshold.

In our experiments, we use a third algorithm, A3. A3 is similar to A1, but does not check the first part of the email, but the last part. We ask all participants in both groups to also break A2 after they have broken A1 and A2. The inclusion of this third algorithm at the end of both experiments does not affect the results for the first two algorithms, as it would not affect the learning cycle on the previous two algorithms, but could enable us to gain valuable insights for future research. The first insight would relate to how the attacker’s increase in knowledge would affect the speed needed to break another related, but not subset, algorithm. The second insight would be a step toward proving that the attacker’s increase in knowledge would affect the speed needed to break another related, but not subset, algorithm. The second insight would be a step toward proving that the attacker’s increase in knowledge would affect the speed needed to break another related, but not subset, algorithm. The second insight would be a step toward proving that the attacker’s increase in knowledge would affect the speed needed to break another related, but not subset, algorithm. The second insight would be a step toward proving that the attacker’s increase in knowledge would affect the speed needed to break another related, but not subset, algorithm. The second insight would be a step toward proving that the attacker’s increase in knowledge would affect the speed needed to break another related, but not subset, algorithm. The second insight would be a step toward proving that the attacker’s increase in knowledge would affect the speed needed to break another related, but not subset, algorithm. The second insight would be a step toward proving that the attacker’s increase in knowledge would affect the speed needed to break another related, but not subset, algorithm.

Algorithm 3 (A3): This algorithm is also similar to A1 except that the hashes are calculated from the last part of the message instead of the first part. We therefore do not provide the pseudo code for A3 separately; it follows directly from the code for A1 in Figure 2.

Table 1. Identifying the symbols and the values

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Spam threshold</td>
<td>100</td>
</tr>
<tr>
<td>N</td>
<td>Number of hash values for each email</td>
<td>100</td>
</tr>
<tr>
<td>S1</td>
<td>Similarity threshold “Algorithm 1”</td>
<td>75%</td>
</tr>
<tr>
<td>S2</td>
<td>Similarity threshold “Algorithm 2”</td>
<td>45%</td>
</tr>
</tbody>
</table>

Briefly, the symbols in Table 1 are explained in the following. As in [17], a hash-based vector representation was used. That is, for each email, hash values of each length 9 substring are calculated1.

1 We use the standard hash function provided in Java library.

and then the first N of them are used as vector representation of the email. To check a single email, in order to find similar previous email which share S% of the same hash values, the algorithm checks the database. As a result, an email transmitted more than D times is marked as spam.

In each trial of the experiment, the participant sends a batch of 100 identical messages to the system, i.e. the number of messages is fixed. In order to pass the algorithm, each participant should be able to get 300 emails accepted by the spam filter. In our experiments, if an email has been sent more than 100 times and share 75%, for example in case A1, of their hash values, they would be marked as spam. So, if a participant sends the same message twice, the system regards it as spam since two hundred identical emails being sent. The participant must thus make enough changes to avoid an email being considered spam. The copy and paste functions were not activated to avoid sending completely different email, and this will be highlighted in the following section. The similarities thresholds were selected empirically, based on the results of the pilot study.

3.1.5 Materials: stimulus and rational

The stimulus material provided to participants consisted of some default email text. The subjects were asked to send this text to the server, as if it was a typical email. The email 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 email text was assigned to all subjects, rather than allowing each subject to write his own email. There were several reasons for this. First, self-written emails may be of different lengths, making the measurement and comparison of participant’s learning a difficult task. Second, self-written emails 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 email 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 email for everyone affected experimental control over unanticipated biases.

3.2 Experimental Procedure

In this section, the instructions to subjects, procedures and collected data are explained.

3.2.1 Instructions to Subjects

As mentioned, subjects were instructed to act as attackers whose target is to defeat the spam-filter algorithms by successfully passing the spam filter for 3 emails (where each e-mail is interpreted as a batch of 100, as we explained above). 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. Subjects were instructed that there are a number of candidate attacks that spammers can enact to fool spam filter algorithms. For example [5, 31]: Random addition, Thesaurus substitution, Perceptive substitution and Aimed addition. 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
emails 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.

3.2.2 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 (20). 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 consumed to complete the task.

During the experiment, each participant was given a brief introduction to the content-based spam-filter. Printed information was also supplied. Participants were highly encouraged to ask the experimenter any questions especially on how to complete the survey at the end of the experiment. A brief demonstration was then provided to participants on how the prototype system works was shown to the participants. This demonstration was performed in a uniform manner across all participants. The participants were then allowed to get a brief hands-on experience using the prototype system. The experimenter remained seated throughout. The details of the experimental task that a participant carried out are the following:

- The participant attempted to defeat the spam-filter by sending a default email 900 times.
- After each sending, the participant progress will be given whether s/he passes or fails, and whether the currently deployed algorithm of the system has been changed or not.
- Finally, the participant was asked to fill a short survey about the previous session.

At the end of the experiment, each participant was informed about the achieved score, the time taken and the number of trials. Each participant also had the choice to fill in a short survey about his or her experience.

3.2.3 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 emails sent for each session were recorded as well.

4. RESULTS AND ANALYSIS
In the experimental study, all the participants successfully completed their task. The details of the overall time needed to defeat all algorithms, the significance of the order and how the attacking process went through are now presented.

4.1 Overall Time
The average time needed to break each algorithm in the two groups is shown in Figure 4. Remember from Section 3.1.1 that Group 1 starts with trying to break the easier algorithm A1 and then the harder algorithm A2, while Group 2 starts with the harder algorithm A2, followed by A1. Both groups conclude with algorithm A3.

From the totals (the right most bar), we see that Group 1 took longer than Group 2. This demonstrates there are implications to the ordering of the algorithm, as we will discuss in detail in Section 4.2. As expected, the ‘tougher’ algorithm A2 took more time to break than A1. In Group 2 it took the longest time, 16.2 minutes, followed by 14.10 minutes in Group 1. Some other results are also as expected. The time needed to break A1 is far less in Group 2, because it learns from first breaking A2, which is effectively a superset of A1. Also, the time needed to break A3 was almost identical in both groups.

![Figure 4. The average time (in minutes) for ‘attackers’ in each group to break the algorithms.](image)

4.2 Order Significance
In the following, the consequences of changing the order of release of the first two algorithms A1 and A2 are presented.

Table 2 compares with respect to the order of the first two algorithms in the two groups, both with respect to time and trials needed. In the tables, ‘Avg.’ denotes average and ‘SD’ denotes standard deviation of the observations. ‘Max’ and ‘Min’ obviously refer to the maximum and minimum of the observations.

The average time needed for breaking A1 and A2 in Group 1 is 25.0 minutes, and 20.10 minutes in Group 2. A t-test yields a result of t=1.89, p<0.1, indicating that the average time needed to break the algorithms was significantly longer in Group 1 than in Group 2.

The order of the algorithms did also influence the average number of trials as it was 33.4 trials for Group 1 and 26.1 trials for Group 2. However, this was not to a statistically significant level (t=1.51, p=0.143).

Table 2. Order matters of two algorithms A1 and A2

<table>
<thead>
<tr>
<th>Group</th>
<th>Total time (A1 + A2)</th>
<th>Total trials (A1 + A2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg.</td>
<td>SD</td>
</tr>
<tr>
<td>1</td>
<td>25.0</td>
<td>10.6</td>
</tr>
<tr>
<td>2</td>
<td>20.1</td>
<td>4.3</td>
</tr>
</tbody>
</table>

We compared A1 in the two groups in Table 3. A t-test yielded a result of t=6.33, p<0.001, indicating that the time consumed to break A1 in Group 1 was significantly longer than that in Group 2. Also a statistically significant difference was found in the number of trials (t=6.62, p<0.005), showing that the number of trials increased correspondingly with the time needed.

Table 3. Breaking A1 for each group

<table>
<thead>
<tr>
<th>Group</th>
<th>Total time A1</th>
<th>Total trials A1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg.</td>
<td>SD</td>
</tr>
<tr>
<td>1</td>
<td>10.9</td>
<td>4.7</td>
</tr>
<tr>
<td>2</td>
<td>3.8</td>
<td>1.3</td>
</tr>
</tbody>
</table>
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 was not significantly longer than that in Group 1. Moreover, there was not found any statistic significantly in the number of trials ($t=0.86$, $p=0.399$).

### Table 4. Breaking A2 for each group

<table>
<thead>
<tr>
<th>Group</th>
<th>Total time A2</th>
<th>Total trials A2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg.</td>
<td>SD</td>
</tr>
<tr>
<td>1</td>
<td>14.1</td>
<td>6.2</td>
</tr>
<tr>
<td>2</td>
<td>16.2</td>
<td>3.4</td>
</tr>
</tbody>
</table>

By examining Tables 3 and 4, two interesting patterns can be observed. First, the attackers gained knowledge after exploiting the rules of A2 in Group 2 was almost enough to enable the attacker to defeat A1 in no time. That is, the time to break the second released algorithm was extensively decreased in Group 2 compared to Group 1. Second, the attackers gained knowledge after exploiting A1 in Group 1 did not contribute much to break the tougher next algorithm A2.

It can also be observed that there is an important variation in the time needed to break A1, which constituted 44% of the time in Group 1 compared to 19% in Group 2. Also, the time needed to break A2 was 56% of the average time in Group 1 compared to 81% in Group 2. As a consequence, this may imply a direct correlation between the order of the algorithms release and their type.

We examined the third algorithm, which is less related to the algorithms A1 and A2, to check whether the algorithm order would have any effect on the time needed to defeat it. That is, we compared A3 in the two groups in Table 5. A t-test yielded a result of $t=0.14$, $p=0.891$, indicating that there was no statistically significant difference between the times needed to break A3 in both Group 1 and Group 2. Moreover, no statistically significant difference was found in the number of trials ($t=1.12$, $p=0.273$). As a result, the order of the preceding algorithms A1 and A2 did not influence the learning phase for the non-subset algorithm.

### Table 5. Breaking A3 for each group

<table>
<thead>
<tr>
<th>Group</th>
<th>Total time A3</th>
<th>Total trials A3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg.</td>
<td>SD</td>
</tr>
<tr>
<td>1</td>
<td>6.70</td>
<td>4.93</td>
</tr>
<tr>
<td>2</td>
<td>6.5</td>
<td>4.4</td>
</tr>
</tbody>
</table>

Finally, we compared the influence of ordering for all the algorithms in the two groups in Table 6. The average time needed for the total of all algorithms in Group 1 was 31.7 minutes compared to 26.60 minutes in Group 2. A t-test yielded a result of $t=1.76$, $p<0.1$, indicating that the time needed to break the algorithms in Group 1 was significantly longer than in Group 2. Interestingly, the average number of trials was 42.3 trails in Group 1 compared to 32.1 trails in Group 2. A t-test yielded a result of $t=1.78$, $p<0.1$, indicating that the number of trials in Group 1 was statistically significantly higher than in Group 2. This is surprising, since the time to break A3 differs little between groups, and without algorithm A3 there was no statistically significant difference between the total number of trials of the two groups. This may suggest that with respect to the number of trials needed the validity of our hypothesis is at the edge of statistical significance.

### Table 6. Breaking all algorithms for each group

<table>
<thead>
<tr>
<th>Group</th>
<th>Total time (A1+A2+A3)</th>
<th>Total trials (A1+A2+A3)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg.</td>
<td>SD</td>
</tr>
<tr>
<td>1</td>
<td>31.7</td>
<td>10.8</td>
</tr>
<tr>
<td>2</td>
<td>26.6</td>
<td>7.36</td>
</tr>
</tbody>
</table>

So, it could be confirmed that concatenating the algorithms (i.e. A1 and A2) in either groups with a different algorithm expands the time needed to defeat the spam-filter regardless of their release order. At the same time, it signifies that once knowledge has been gained, the success in breaking other algorithms does not depend on how that total amount of knowledge was gained. This is perhaps as expected, but it is in fact important in our Markov model in Section 5, which by the nature of Markovian processes requires such a ‘memory-less’ property.

### 4.3 Attacking Process

Qualitative data were collected, in the form of surveys, to verify that the attacking process was accomplished by structured strategies that are based on the knowledge gained rather than complete randomness. In particular, we were looking for the strategies that were used to defeat the algorithms, the part of the email that the participants believed that each algorithm was checking, and the algorithm which the participants thought was the toughest to defeat.

Most of the participants, 90% (36 out of 40) used structured strategies to defeat the algorithms. In particular, 55% (22 out of 40) used random addition, Thesaurus substitution and perceptive substitution, while the remaining 35% (14 out of 40) used Thesaurus substitution and other strategies.

In terms of identifying the correct part of the email that each algorithm is checking, the results in Group 1 were 70% (14 out of 20), 80% (16 out of 20) and 75% (15 out of 20) for A1, A2 and A3, respectively. In Group 2, the results were 100% (20 out of 20), 100% (20 out of 20) and 70% (14 out of 20) for A1, A2 and A3, respectively. Furthermore, we observed that 80% (16 out of 20) in Group 1 found that the A2 was the hardest algorithm, whereas 95% (19 out of 20) found that the A2 was the hardest algorithm in Group 2.

It is worthwhile to note that pervious research assumed, based on empirical results, that the attackers’ skills would increase based on the knowledge acquired [6]. Our qualitative data appeared to confirm this assumption.

The experiment shows that there is a statistically significant difference in the time taken by the two groups to break the algorithms. In other words, the hypothesis (H1) holds that the total time for an attack to succeed depends on the order in which defensive algorithms are released (or, to be more precise, the hypothesis is not rejected). Given this result, we seek to construct a model through which the optimal release strategy for defense mechanisms can be derived.

### 5. DERIVING OPTIMAL RELEASE STRATEGIES

To determine the optimal release strategies we use a stochastic model that takes into account the important aspects of the problem. Broadly, a stochastic model is a model that involves probability, or randomness, associated with time and events. When using such a model, a stochastic process represents the...
behavior of the system over time, and given the occurrence of certain events. A stochastic model can be depicted as a state transition diagram, which describes all relevant operational system states and the possible transitions between these states. To describe time aspects between events, a rate matrix is specified. One usually assumes that the event that will occur next in the system, as well as the time before this event, is random. Hence, the behavior of the system is a stochastic process. The main advantage of this modeling approach is that it captures dynamic aspects of system behavior, which we argue is an applicable approach for modeling the security of a system.

In particular, we will model the problem as a Markov Decision Process [3], allowing us to determine an optimal strategy from the model. The objective will be to order the release of algorithms such as to maximize the time until the attacker breaks through all algorithms available. We will show that when maximizing the mean time to break the algorithms, well-known iterative algorithms [16] can be applied; moreover, the problem is a special case of [7]. We will provide a version of such an iterative algorithm that exploits the absorbing nature of the underlying Markov chain and avoids generating (and storing) the whole Markov chain.

5.1 Markov Decision Process

A decision process is characterized by the fact that in each state there is a choice to be made between possible actions. Each action takes the process to a new state. For the problem at hand, we introduce a continuous time Markov Decision Process defined by its states \( s \in S \), the possible actions \( A(s) \) in any state \( s \), and transition delays \( \lambda_{i,j} \) associated with any action \( a \in A(s) \). If useful, we will also use \( \lambda_{i,j} \) if the action results in a transition from state \( i \) to \( j \).

States in the Markov Decision Process must reflect the amount of knowledge gained by the attacker. We make the very natural assumption that if a given set of algorithms has been broken, the time it takes to break future algorithms is the same regardless of the order in which the earlier algorithms were broken. Then, the state is completely specified by maintaining which algorithms are broken. If \( G \) is the set of all algorithms (with \( |G| \) elements), then a state \( s \in S \) is a tuple \( s = (g_1, g_2, ..., g_{|G|}) \), where \( g_i = 0 \) if the \( i \)-th algorithm has not been broken yet, \( g_i = 1 \) if the \( i \)-th algorithm has been broken.

The actions in a state represent the selection of a next algorithm to be released. So, there is an action corresponding to any algorithm that is not broken yet, that is, there are as many possible actions in state \( s \in S \) as there are 0 elements.

The delays signify the time it takes for an attacker to break the algorithm associated with action \( a \). This time depends on the knowledge gained from breaking earlier algorithms, which is maintained in the state.

This formulation immediately shows that there exist at most \( 2^{|G|} \) states. The possible order in which the \( |G| \) algorithms can be released is \( |G|! \). To determine which release order is optimal, we first need to define the optimization criterion. For that optimization criterion, we then require a reasonably efficient algorithm to search through the many options.

The metric of interest (and, hence, the optimization criterion) in our work is to maximize the time it takes to break all algorithms.

So, let the stochastic process \( R(t) \), defined for \( t \geq 0 \), indicate if all algorithms have been broken at time \( t \): \( R(t) = 1 \) if \( s = (1,1,...,1) \) and otherwise \( R(t) = 0 \). Note that \( R(t) \) turns 1 only once, and then stays 1. The probability that all algorithms are broken at time \( t \) is \( P(R(t) = 1) \), where \( P \) indicates the probability, as usual. \( R(t) \) also provides us with the mean time to security failure (MTTSF [15]):

\[
E[R(t)] = \int_0^{\infty} (1 - R(s)) \, ds,
\]

and with higher moments similarly. In what follows, we refer to \( R(t) \) as the time to security failure.

Finding the best strategy corresponds to a standard Markov Decision Process optimization problem with finite horizon only for the first moment \( E[R(t)] \), but not for higher moments or its distribution. We will also present a specific backward algorithm that efficiently generates all paths ‘backwards’ from the state that all algorithms have been broken.

5.2 Optimization Algorithm

To calculate the optimal strategy, it is useful to realize that any selected sequence of algorithms corresponds to a hypo-exponential distribution, which in turn is a special case of a Phase-Type distribution [11]. We now need the following result for hypo-exponential distributions:

If \( H_{i} \) is hypo-exponential with rates \( \lambda_{1},...,\lambda_{K} \) and MTTSF \( E[R_{i}(t)] \), and \( H_{0} \) is hypo-exponential with rates \( \lambda_{0}, \lambda_{1},...,\lambda_{K} \), and MTTSF \( E[R_{0}(t)] \), then \( E[R_{0}(t)] = 1/\lambda_{0} + E[R_{i}(t)] \).

This is an obvious result, but it is important to note that the same does not hold for higher moments. The above implies that we can execute a backward algorithm that optimizes for hypo-exponential distributions of increasing length. It also implies that we can use known Markov Decision Process theory, since we can associate reward \( r_{i,j} = 1/\lambda_{i,j} \) with each transition from state \( i \) to \( j \). The association of rewards completes the formal definition of a Markov Decision Process.

Because of the specific structure of our model, it makes sense to provide a bespoke algorithm that avoids generating the complete state space as shown in Figure 5. Note that \( (1,1,...,1) \) is the absorbing state with all algorithms broken. The algorithm starts from that absorbing state and explores all possible previous states (stored in \(ToDoSet \)). For each previous state it selects the action that maximizes the time to reach the absorbing state (stored in the \(BestNext\) variable associated with each state). This continues until the state with no broken algorithms is reached.

```start = (0,0,...,0);
end = (1,1,...,1);
For All s ∈ S set ET_s = 0;
ToDoSet = {end};
While( ToDoSet ≠ {start} ) Do {
    ToDoSet = {s|s→i, for any i ∈ ToDoSet} 
    For All s ∈ ToDoSet Do {
        For All i ∈ S such that s → i Do {
            If( 1/λ_s,i + ET_i > ET_s ) Then {
                ET_s = 1/λ_s,i + ET_i;
                BestNext_s = i;
            }
        }
    }
}
```

Figure 5. The backward optimization algorithm
The optimal order of releasing the algorithms is then obtained as follows, in the tuple Optimal:

```plaintext
s = start;
Optimal = (s);
While( s ≠ end ) Do {
    Optimal = (Optimal, BestNext_s);
    s = BestNext_s;
}
```

We note that the algorithm above does not generate the complete state space S, nor all possible sequences of algorithms. The storage required is about \(N!/[(N/2)!(N/2)!]\) real-valued variables, which occurs halfway the backward algorithm (which starts with a single state (end) and ends with a single state (start)). That still limits the size of the model one will be able to solve, but with modern day computing equipment this implies that the problem can be solved for up to several tens of algorithms.

It is important to note that the above algorithm does not work if higher moments are considered. Moreover, it is also straightforward to find release strategies that optimize the MTTSF, but not optimize the second moment of the time until security failure.

5.3 Application to the Example

Our example with the three algorithms is of course a simple case, in that it has only few states, and the best release strategy can therefore be easily computed. Nevertheless, it is useful to provide the Markov Decision Process for this case, as we do in Figure 6.

![Figure 6. Markov Decision Process for Example](image)

In our example there are three algorithms, \((A1, A2, A3)\), leading to 8 theoretically possible states (denoted by the circles in Figure 6), but in our example we restrict the possible order and always put algorithm A3 last. The actions in each state are given by the arcs. Only in state \((0,0,0)\) there is a choice between actions, namely to first release algorithm A1 (leading to \((1,0,0)\)) or to first release algorithm A2 (leading to \((0,1,0)\)). The arcs are labeled by the time it takes to complete breaking the algorithm, as seen from our experiment. Referring back to Section 4, Group 1 followed the trajectory at the top of Figure 6, using 10.9 minutes to break A1 and 14.1 to break A2. Group 2 followed the trajectory at the bottom of Figure 6, using 16.2 minutes to break A2 and 3.8 minutes to break A1. Then all participants broke A3, in an average of 6.6 minutes.

The backward optimization algorithm of Figure 5 would traverse backward and pick the best action. Before getting to state \((0,0,0)\), it would have obtained intermediate results of 14.1 + 6.6 = 20.7 for state \((1,0,0)\) and 3.8 + 6.6 = 10.4 for state \((0,1,0)\). For state \((0,0,0)\) it then selects the action that maximizes the time to security failure, so it would release algorithm A1 first (the trajectory at the top of Figure 6), because 10.9 + 20.7 > 16.2 + 10.4. So, the optimal release strategy becomes A1 followed by A2 followed by A3.

6. DISCUSSION

Our experimental and theoretical study confirms that optimizing the release order for the same set of algorithms can increase the time needed to break a system’s security in a statistically significant manner.

In particular, as shown in Table 2, the time and number of trials that were required to break the system in Group 1 were more on average compared to Group 2. We remind the reader that algorithm A1 is a subset of A2, which implies that when A2 is broken, the same technique would break A1. For this particular experiment, we can conclude 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 that the intuitive reasoning that by breaking up an algorithm in subsets you ‘teach’ the attacker how to attack is less valid.

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 email. In contrast to our results, a study in [29] shown that the low-variance condition group performed better than high-variance condition group with regards to accuracy. However, they did not evaluate whether or not the order of low/high-variance can be performed differently.

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. It is still true that the time needed to learn was identical in both groups and less than the first learning phase. 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.

The results obtained in this paper are very encouraging. They indicate that the order in which defensive algorithms are released matter for the example and may therefore matter in various settings. Hence, optimizing the release order of defensive algorithms is a problem worthwhile to be studied. The results in this paper are a first step, showing the validity of the problem and providing insights in where to invest future research.

7. RELATED WORK

A considerable number of studies have been conducted by researchers to investigate the protection of systems from attacks by automated software that degrade their quality of service due to resource expenditure.

**Attack Modelling:** A quantitative analysis of attacker behaviour based on empirical data collected from intrusion experiments was presented in [6]. Beside this, [12] has described a technique for transforming a privilege graph into a Markov chain. 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 and allow the analysis of the security of systems capable of detecting and responding to attacks [8, 1]. In relation to the time taken for an attacker to compromise a system and misuse its resources, [9, 10, 28, 32] have proposed a model for estimating the time to compromise a system component that is visible to an attacker. Additionally, [10] suggested the attacker skill levels should be considered when determining the mean time to compromise a system [9]. The studies listed above suggest that stochastic modelling can be a suitable approach to the modelling of attacks on a system, and quantifying the efficacy of security measures. While a number of the above methods and techniques seem promising, none of them could provide a quantitative measure of maximising the time needed to defeat a system that was considered for the proposed approach.

Game theoretic Security Approaches. Game theory has been a mainstream research topic in the economic community. A comprehensive introduction to the area is in [22]. Applied economics concepts have been applied to computer security to address the analysis of strategic choices that enterprises will take regarding to maintenance and management under an assumed cost. As game theory views the interactions between an attacker and the administrator as a two-player, it can provide a mathematical framework for analyzing and modeling network security problem. As indicated in [33], most of the current games theoretic are based on static game, games with perfect information or games with complete information.

In the traditional network security solutions, one of the first approaches for applying game theory to network security is discussed in [20]. They used a Markovian decision process and one player game to detect, reason, and respond to automated attack behavior in information assurance systems. Furthermore, Lye and Wing [25] model the interaction between an attacker and a defender as a two-player stochastic game. In P2P Session Initiation Protocol infrastructure, game theory is used to understand and better defend against blocking and flooding attacks against [19].

As a dynamic game, the problem of Nash Equilibrium Design for quite a general class of games from an optimization and control theoretic perspective investigated by Alpcan et al. [18]. They focused on how long does the game approach Nash equilibrium when many players are trying to solve it in a distributed way. A feedback system approach is suggested as a control input to make the system robust and to control the system’s progress. [26] utilized Min-max Q learning approach to aid in the gradual improvement of the defender’s quality. This work can handle reactive defense actions, whereas a few others did not consider a realistic attack scenario. More importantly, 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. General learning theories, such as Rumelhart and Normans [30], model of accretion, tuning and restructuring. For example, an empirical non-linguistic experiment by [29] evaluated 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. Our focus on attackers and defensive algorithms is clearly different, but when studying the learning process of attackers in more detail, undoubtedly lessons can be learnt from [30] and other literature in the field of education.

8. **CONCLUSION AND FUTURE WORK**

This paper introduces the problem of scheduling the release of defensive algorithms so as to successfully defeat all algorithms. 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 an 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.

The paper also provided a general approach to compute the optimal release strategy for defensive algorithms. We mathematically model the problem as a Markov Decision Process and provide a tailored algorithm to efficiently solve any model within the class of models presented. The model solution should scale without problems to optimize the release order of tens of defensive algorithms.

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. Other fields, such as education and psychology, may provide a basis for such research. 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. We look forward to investigate such problems further.

9. **ACKNOWLEDGMENTS**

Removed to facilitate blind review.
10. REFERENCES


[33] S. Roy, C. Ellis, S. Shiva, D. Dasgupta, V. Shandilya, Q. Wu. 2010. A survey of game theory as applied to network...