Making It Easier To Publish Your Work

***Abstract*—With an increasingly high number of submissions and reviewers to the top research conferences, the assignment of papers to reviewers is a difficult problem facing conference organizers. In this paper, we show that it is possible to modify a paper submission in such a way as to force the Toronto Paper Matching System (TPMS), a popular solution to this problem, to pick or avoid arbitrarily chosen reviewers for the paper. Specifically, we embed keywords into the PDF so that there are no visible changes to the document but TPMS determines the paper is related to a desired reviewer's area of expertise or unrelated to an unwanted reviewer. We show the effectiveness of our attack through various experiments, and find that an attacker can get her paper reviewed by a favorable reviewer with a probability as high as 80% and can avoid unfavorable reviewers with a probability as high as 99%. We also highlight potential mitigations against the proposed attack and exciting directions for future research in this area.**

I. Introduction

One of the most important and difficult tasks facing an academic conference organizing committee is the assignment of papers to reviewers. Top conferences receive thousands of submissions, which have to be assigned to thousands of reviewers (NeurIPS 2019 had 6743 submissions [8] and more than 4000 reviewers [1]) within a few days after the conference submission deadline. Moreover, the assignment of each paper requires knowledge of both the topics explored in the paper as well as knowledge of reviewers' expertise. Other additional constraints, such as reviewer load, make the task impossible to be done manually by the program chair. This scale also makes it difficult for reviewers to rank all the submitted papers in the order of their preference. Hence, most conferences now use partial ordering by the reviewers and an automated system for this task.

The Toronto Paper Matching System (TPMS) [7] is one such popular system that has been used repeatedly by NeurIPS, ICML, UAI, CVPR, ECML/PKDD, etc. The system first determines reviewers' expertise by looking at their own published papers, either crawling through their Google Scholar profile[1] or using papers submitted by the reviewers. TPMS then extracts features (bag of words) from the PDFs of papers submitted to the conference. A machine learning model, assisted by the partial rankings provided by reviewers, is learned to score each paper-reviewer pair based on the similarity of a paper's features to a reviewer's expertise. For this project we work with AutoBid, an open source implementation of TPMS [11].

[1]http://scholar.google.com

Network and Distributed Systems Security (NDSS) Symposium 2021
21-24 February 2021, San Diego, CA, USA
ISBN XXXXX
https://dx.doi.org/10.14722/ndss.2021.24xxx
www.ndss-symposium.org

TABLE I: Key dataset statistics.

# of submissions	3188
Total # of reviewers	2726
Total # of reviewers' publications	15225
# of reveiwers with at least 1 publication	2669
# of reviewers with at least 5 publications	1759

We show that it is possible to trick TPMS into both assigning an arbitrary favorable reviewer to a paper and avoiding an unfavorable reviewer by embedding metadata into the submission PDF. Specifically, we achieved two major goals with this project. We automate the process of embedding arbitrary text into any PDF such that the embedded text is parsed by command-line tools but not rendered by GUI-based PDF viewers. Thereafter, we come up with specific adversarial keywords that trick TPMS into selecting or avoiding a reviewer of the adversary's choice for any given arbitrary paper. Putting both of these goals together, we alter a conference submission, without causing any visible side-effects, in a way that causes TPMS to behave as we desire.

II. Goals

With a dataset of size comparable to that of a top conference, we designed our attack on TPMS with the following goals:

1) Embed keywords in a paper PDF so that there are no visible changes to the PDF, but the keywords are read by PDF parsers.
2) Force TPMS to select or avoid a specific reviewer for a modified paper.
3) Automate the process for arbitrary reviewers and arbitrary papers.

III. Dataset

In order to come up with a reliable way to trick TPMS, we need access to an archive of submitted papers, a list of sample reviewers and the reviewers' representative publications. The scale of the dataset should be comparable to a real top conference for us to make conclusive statements about our methods. For this project, we focus on the data from NeurIPS 2015-2019 [2].

Table I lists the key statistics of our dataset. Because the list of all submissions to a conference is generally not made public, we collect the papers accepted in NeurIPS in the past five years as our set of submitted papers [3]. We have a total of 3188 papers as our set of submissions.

To compile our reviewers, we use the publicly available list of NeurIPS 2019 reviewers [1]. Although the web-page lists a total of 5133 reviewers, only 2726 of these are uniquely

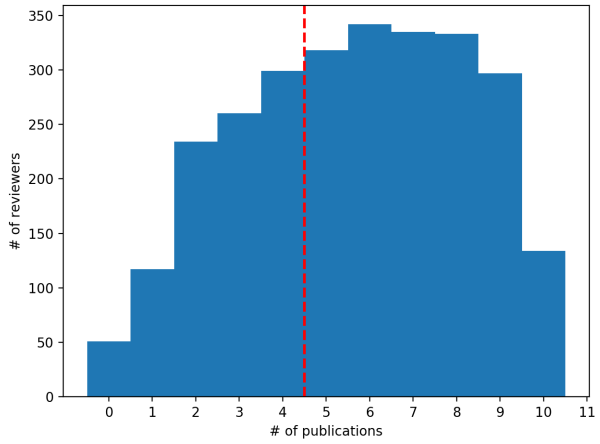


Fig. 1: Distribution of the number of fetched publications per reviewer. The red line indicates the threshold of number of publications per reviewer. We use only the reviewers to the right of this threshold.

identifiable by a Google Scholar search, which is necessary to find their publications. Fig 1 shows the distribution of the number of fetched publications per reviewer. Out of a total of 2726 reviewers, we have at least one publication for 2669 with an average of 5.6 ± 2.6 publications per reviewer. TPMS recommends using reviewers with at least 5 publication PDFs. We have 1759 reviewers that fit this criteria, which entails about 5.4 reviews per reviewer, assuming that each submission is reviewed by 3 reviewers.

IV. Running TPMS on our dataset

As previously stated, we use the open source implementation of TPMS known as Autobid [Parno 2020]. The Autobid repository contains two main branches: the `master` branch and the `prior_bids` branch. We worked with the `master` branch for this project. The two branches differ in how they create their models of reviewer expertise but not in how they scrape the PDF files. The `master` branch uses an LDA model for generating bids while the `prior_bids` branch uses StarSpace [14]. If our work was extended to the `prior_bids` branch, our method of embedding key words into the PDF would not need to be changed. However, the method of selecting key words to embed would need to be changed to fit the `prior_bid`'s reviewer expertise model.

On the `master` branch, Autobid has four main steps to generate reviewer-submission matchings. First, the reviewer publications are analyzed. Second, the submitted papers are analyzed. Third, both the reviewer publications and the submitted papers are used together to build a topic model mapping word occurrences to topics. Finally, the analysis of the reviewer publications, the analysis of the submitted papers and the topic model are all used together to generate the final matchings.

V. Threat Model

We assume that the attacker submits a paper for publication in one of the top conferences and intends to have her paper either reviewed by a favorable reviewer or avoid an unfavorable reviewer. We assume that the attacker has access to the papers selected for publication in the previous editions of the conference, and that the attacker has the list of names of the reviewers set to participate in the upcoming edition. Given that many conferences make their previous publications and the list of reviewers public before the deadline of paper submission, we believe this is a reasonable and realistic assumption. The attacker only controls her own submission to the conference, which is in the form of a PDF. We adopt two distinct threat models. First, we use a restrictive threat model where the attacker can only add some hidden words to the PDF without modifying the existing content. Second, we use a more realistic threat model in which the attacker can make the existing content ignored by Autobid while also adding hidden words. In real scenarios, the attacker can tailor the actual content of her submission to be adversarial as well, but we do not consider that case here. In our restrictive settings, we divide our task into two parts. First, we create a module to embed a given list of arbitrary words in a given PDF without altering the rendered view of the PDF. Second, we come up with a list of adversarial keywords that need to be hidden in the submission. The next two sections describe these tasks in detail.

VI. Modifying PDFs

Autobid uses the `pdftotext` [6] utility for parsing text from PDF submissions. To trick Autobid, we need to modify a submission in such a way that `pdftotext` registers our changes, but the PDF remains the same visually. We break this task into two parts: making `pdftotext` ignore the original text of the submission and embedding new text into the submission.

A. Ignoring original text

There are a number of ways to take advantage of the PDF format to make text that is not extractable by parsers but still renders normally. For example, the text can be displayed using vectors drawn in the shape of individual letters. However, many of these methods would be cumbersome to work with, especially when editing existing PDFs rather than creating an adversarial PDF from scratch. Our approach used the `ToUnicode` table, which maps PDF character codes to Unicode values. We first add a new object to the PDF that contains a `ToUnicode` table mapping all character codes to the space character. We then use a modified version of Didier Stevens' PDF parser [12] to find and edit all Font objects in the original PDF. We add a `ToUnicode` attribute to the Font objects and point it to the new table we have created. This causes `pdftotext` to reference our `ToUnicode` table for all text in the document. All text is then extracted as a space character, which Autobid ignores.

B. Embedding new text

When embedding new text, we must ensure that `pdftotext` extracts the new text. Although Autobid does not currently use them, `pdtotext` has a number of options to exclude text that is not readily visible, such as text outside the content bounding boxes, text drawn in PDF render mode 3 (invisible), and text with opacity less than 0.001. We therefore wanted to avoid embedding our keywords using these methods,

as the developers of Autobid could mitigate our attack simply by enabling these extra options when running `pdftotext`.

We decided instead to simply color our embedded text white. As it is not drawn in render mode 3 and has full opacity, `pdftotext` will not remove our keywords unless they are outside of the content bounding boxes. To keep our keywords inside the bounding box, we overlay them with the original content of the PDF. Of course, our keywords must go beneath the original content when overlaying in order for the modified document to appear the same as the original. Placing the embedded keywords behind the original words has an added benefit in that the embedding is more robust to scrutiny under text selection. If someone reading the modified document highlights a portion of the document containing our embedded text, it will appear as if they are highlighting the original text above it. The reader will need to copy their selection and paste it somewhere else in order to see the embedded text.

In order to perform the overlay, we first create a PDF containing only the adversarial keywords. We call this PDF the background PDF. The text in the background PDF is white, so each page appears blank. We use the open-source library ReportLab [5] to generate the background PDF. Once we have the background PDF, we use the open-source library PyPDF4 [4] to overlay each page of the original PDF on top of the corresponding page in the background PDF. If the background PDF has more pages than the original PDF, then the keywords contained in the extra pages of the background PDF will not be present in the final modified document. However, the background PDF almost always has less pages than the original PDF. This is because the background PDF does not have large margins, tables, figures, etc., and therefore compacts the text more tightly than the original PDF. In the case that the background PDF has less pages, the extra pages on the original PDF are unchanged in the final adversarial document.

We initially tried to embed the adversarial words by adding new objects to the PDF file and hacking its 'xref' table to include the injected object in the PDF's object hierarchy. However, that approach turned out to be too complex and fragile, and hence we decided to continue with this simpler method of overlaying the PDFs. Using the two previously mentioned libraries to generate our modified PDFs greatly simplified our implementation: creating an adversarial PDF given a list of keywords required only 67 lines of Python code. However, the trade off for this simplicity is that our tampering is easier to detect. Above, we stated that highlighting the document would not reveal the white text. Actually, this is true only if the white text is behind regular text. Our implementation does not take the layout of the original document into account and therefore places white text in locations where nothing is above it. If a reader highlights these sections of the document, they will notice that they are selecting text that seemingly does not exist and therefore will likely discover the embedded text. This problem could be fixed by writing an embedder that writes white text only to locations containing text in the original document.

VII. Choosing keywords to embed

Autobid builds an LDA model using a provided corpus of representative publications to learn 'topics.' As mentioned previously, we use the combined set of submissions and reviewers' publications as our corpus. Each topic is associated with a list of words and a probability distribution over those words. A high probability indicates that the word is highly indicative of a document belonging to the topic. For each reviewer, Autobid extracts the list of representative topics using the words present in all of the reviewer's publications. Similarly, Autobid extracts the list of representative topics for each submission. Let p_r^t be the probability that reviewer r has expertise in topic t, and p_s^t be the probability that submission s belongs to the topic t. The bid by reviewer r for submission s, B_{rs} is calculated as $B_{rs} = \sum_{t \in T^s} p_r^t \times p_s^t$, where T^s is the set of all representative topics for submission s. The bids for each reviewer are then normalized such that the bids are integers in $[-90, 100]$.

Our goal is to alter a submission s such that the bid by a selected reviewer r for that submission is greater than that by any other reviewer. Hence, we hope to make our modified submission mirror the topic distribution of the favorable reviewer. For each of the top n_t topics of the reviewer, we take the top n_w words and generate a score for each distinct word. The score for word w is generated as the maximum probability of the reviewer's topic in which the word occurs. Formally, $s_w = max_{t \in T_w^r}(p_r^t)$, where T_w^r is the list of topics in which reviewer r has expertise and word w is indicative of the presence of the topic. These scores are then normalized over all words to convert it into a probability distribution.

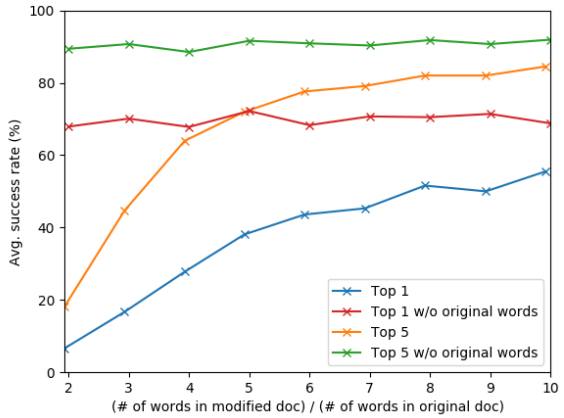
We also tried several other ways to generate scores for each word, such as weighting the generated score by the probability of the word contributing to the topic, weighting rare topics more than those found frequently, etc. However, we found that these additional weights on the score for each word did not help with the performance. We also tried various values for n_t and n_w and found that using top three topics for each reviewer ($n_t = 3$) and top 10 words for each topic ($n_w = 10$) resulted in the best performance. Given an upper limit on the total number of hidden words, a list of words is then generated such that the distribution of words in the modified submission is as close as possible to this adversarial probability distribution.

VIII. Evaluation

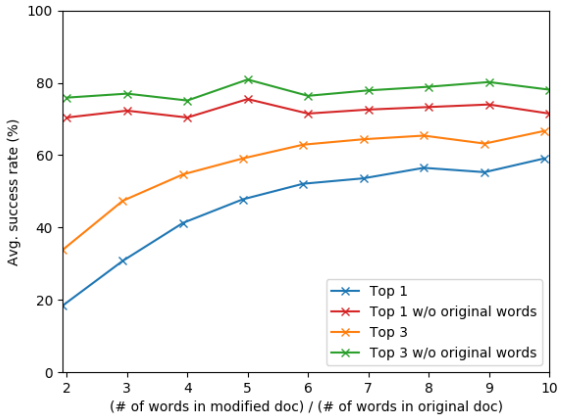
A. Selecting specific reviewers

We evaluate our current attack by using our system to modify a randomly selected submission in order to force Autobid into assigning the submission to a randomly selected favorable reviewer. For all the experiments, we only consider those 1759 reviewers that had at least 5 publications. Moreover, we generate our adversarial words using the top 3 topics for each reviewer and the top 10 words for each topic. We show the average of 1000 iterations for each data point in our results and perform these trials both with the original text of the submission included and excluded from the adversarial submission.

Fig 2 shows the variation in our attack's success rate as the size of the adversarial submission is increased. Fig 2a shows the percentage of trials in which the adversarial submission ranks in the top 1 or top 5 of the favorable reviewer's bids. It must be noted that since we have 3188 submissions and 1759 reviewers, and each submission has to be reviewed by

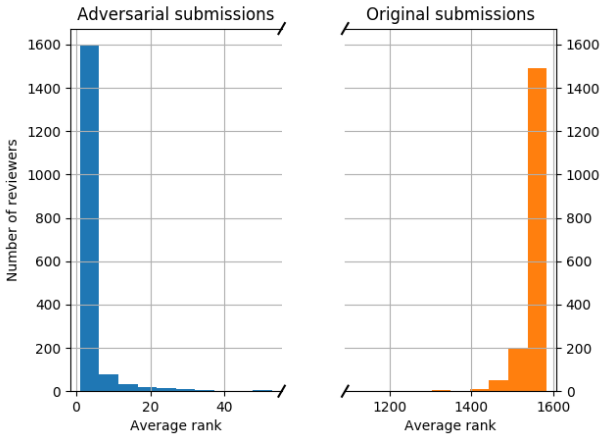


(a) Rank of the adversarial submission in the favorable reviewer's bids.

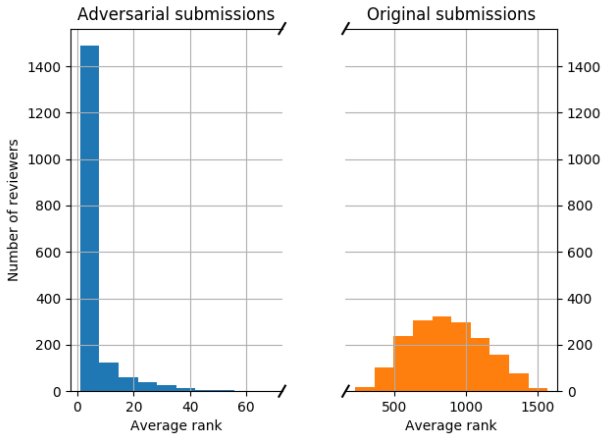


(b) Rank of the favorable reviewer in bids for the adversarial submission.

Fig. 2: Success rate of our attack as the size of the adversarial document is increased. Only one submission is adversarial.



(a) Distribution of the average rank of a submission in bids made by the reviewer.



(b) Distribution of the average rank of a reviewer in bids made for a submission.

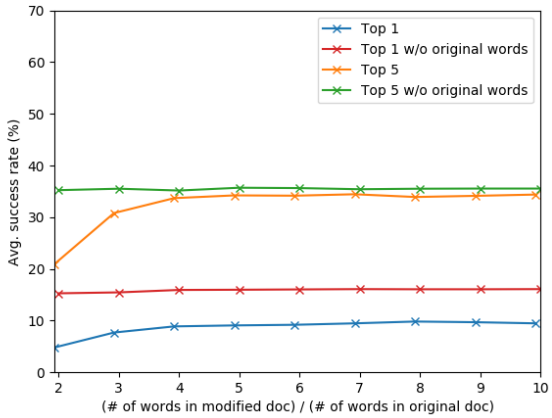
Fig. 3: Bids made by most of the reviewers are vulnerable to adversarial submissions, as shown by our experiment matching every reviewer with every paper (targeting one reviewer for one submission at a time and keeping all other submissions unchanged).

3 reviewers, each reviewer gets to work on 5.4 submissions on average. Therefore, for our attack to be successful it is sufficient that our submission features in the top 5 bids by the reviewer.

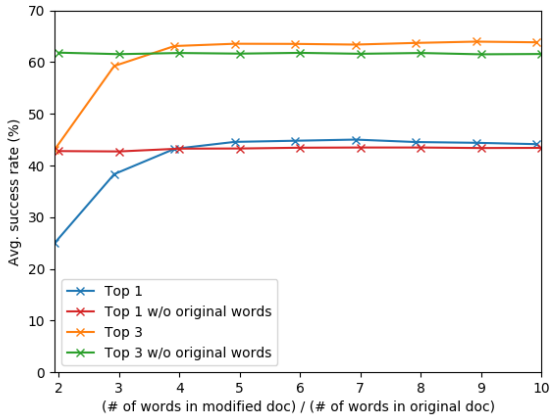
Similarly, fig 2b shows the percentage of trials in which the favorable reviewer ranks in the top 1 or top 3 of the bids for the adversarial submission. Since each submission has to be reviewed by 3 reviewers, for our attack to be successful it is sufficient that the favorable reviewer's bid ranks in the top 3 bids for the adversarial submission.

We can say that our attack is certainly successful when both the submission ranks in the top 5 bids by the reviewer and the reviewer ranks in the top 3 bids for the submission. In our restrictive scenario where the attacker does not modify her paper's original content, the success rate of our attack improves as we include more adversarial words in the document. This is because we are able to offset the original distribution of words and tilt it towards our desired adversarial distribution as we include more and more adversarial words. When the original content is ignored by the text extractor, the success rate remains relatively constant.

With the original text included, we achieve success rate close to 50% when the adversarial document contains nearly 3 times the number of words than in the original document. This success rate becomes close to 70% when we allow the number of words in the modified document to be more than 6 times the number of words in the original document. When the original text is excluded, we achieve a success rate close to

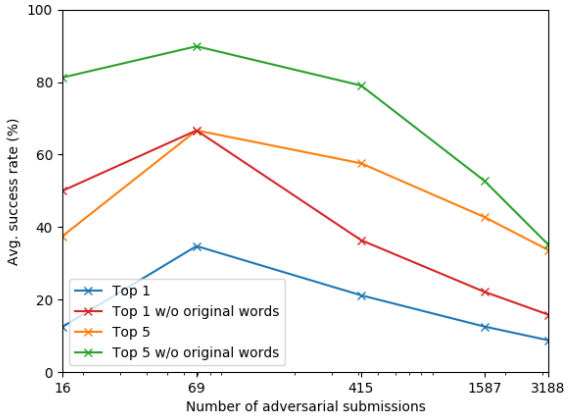


(a) Rank of the adversarial submission in the favorable reviewer's bids.

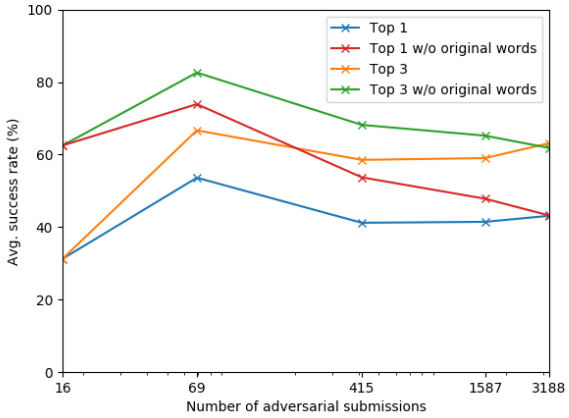


(b) Rank of the favorable reviewer in bids for the adversarial submission.

Fig. 4: Success rate of our attack as the size of the adversarial document is increased. All submissions are adversarial.



(a) Rank of the adversarial submission in the favorable reviewer's bids.



(b) Rank of the favorable reviewer in bids for the adversarial submission.

Fig. 5: Success rate of our attack as the number of adversarial submissions is increased. The number of words in each modified document is fixed at nearly 4 times the number of words in the original document.

80% for all document sizes. To put this in context, the original unmodified document never matches our success criteria in any of the trials.

Additionally, we test our attack by attempting to trick Autobid into matching every reviewer with every paper (selecting one matching at a time). With 1759 reviewers and 3188 submissions, there are approximately 5.6 million trials. For all trials, we exclude the original text of the document. Fig 3a shows the number of reviewers based on the average rank of the matched submission in the list of all bids made by the reviewer. Fig 3b shows the number of reviewers based on the average rank of the reviewer in the list of all bids for the matched submission. On average, 73.9% of reviewers rank in the top 3 reviewers for their matched submissions, while also having the matched submissions rank in the top 5 of their bids. This shows that the majority of reviewers are possible to favor successfully.

B. *Multiple adversarial submissions*

We test the performance of our attack when our threat model is relaxed to allow for multiple (although not necessarily collaborating) adversarial submissions. Fig 4 shows the average success rate when all submissions are adversarial. For this experiment, we choose a desired mapping from each submission to a random reviewer. Since we have almost three times as many submissions as reviewers, this setting unavoidably results in competing adversarial submissions. The dropped success rate of the adversarial submission being in the

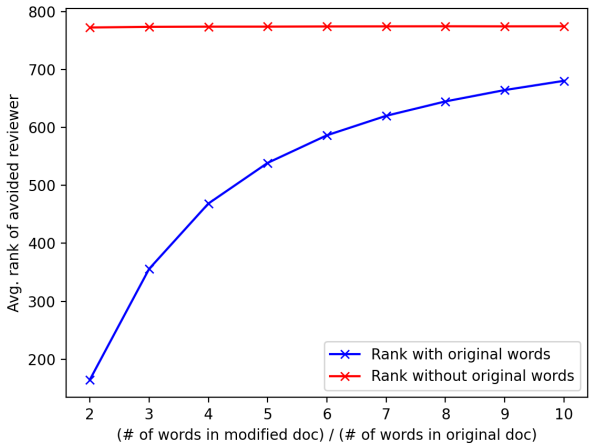


Fig. 6: Rank of the original top reviewer after modifying a submission to avoid that reviewer.

favorable reviewer's top bids is reflective of this competition (Fig 4a). Since bids for each reviewer are normalized to an integer in $[-90, 100]$, higher absolute score by a reviewer for one submission leads to lower normalized bids by that reviewer for other submissions. This leads to lower chance of a favorable reviewer's bid being in the top bids for a submission in presence of competition from other adversarial submissions (Fig 4b). Interestingly, when all submission are adversarial, the performance does not vary greatly with the size of the documents. Overall, nearly 30% of the submissions are assigned to their favorable reviewer as opposed to less than 1% when the submissions are not modified to be adversarial.

We also run an experiment where only a subset of the submissions made to the conference are adversarial. Once again, the submissions are not collaborating and may compete for the same reviewer. Fig 5 shows the success rate as the number of adversarial submissions is increased. As evident from Fig 5a, it is easier for a submission to rank in the reviewer's top bids when there are few adversarial submissions and therefore less competition. On the other hand, as Fig 5b suggests, it is difficult to get the favorable reviewer's bid to be the top bid for an adversarial submission both when there are few adversarial submissions (as it is difficult to exclusively affect bids for one reviewer) and when there are many adversarial submissions (due to competition between submissions for the same reviewer). Hence, the best scenario for the attackers occurs when nearly 100 out of 3188 ($\sim 3\%$) submissions are adversarial. This gives each attacker about 80% chance to succeed as compared to less than 60% when they are the lone attacker. This suggests that even if attackers are not explicitly collaborating, they can still help each other out, as long as there are not too many adversarial submissions. Furthermore, it hints at the potentially disastrous results for a conference if the authors decide to collaborate and make adversarial submissions.

C. Avoiding specific reviewers

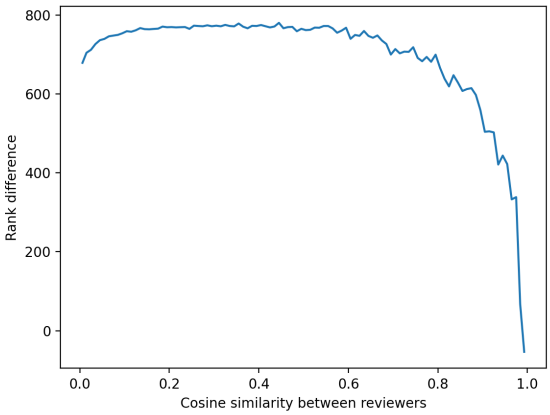
We also evaluate our attack when trying to avoid reviewers rather than selecting them. In this case, we embed keywords by selecting topics that are, on average, more likely for the majority of reviewers than they are for the avoided reviewer. We embed the top 10 words for each such topic, repeated as necessary to achieve the desired ratio between the number of words in the modified submission and the number of words in the original submission. For each word ratio, we perform 3188 trials, one for each submission. In each trial, we modify the submission to avoid its original top reviewer. Our attack performs extremely well, both when the original text of the submission is included and when it is excluded. For all word ratios, the avoided reviewer remains in the top 3 reviewers for the modified submission less than 1 percent of the time. The average rank of the original top reviewer after modification is shown for various word ratios in Fig 6. When the original text of the document is excluded, the average rank of the reviewer remains constant at approximately 775. As expected, when the original text is included, the average rank of the reviewer varies much more with the size of the adversarial submission. However, even with only 2 times the number of words in the original PDF, the average rank of the reviewer is about 165. Our results in these experiments show that it is more reliable for an attacker to try to avoid a reviewer she knows is likely to bid highly on her paper than it is for her to try to favor an arbitrary reviewer.

D. Choosing between similar reviewers

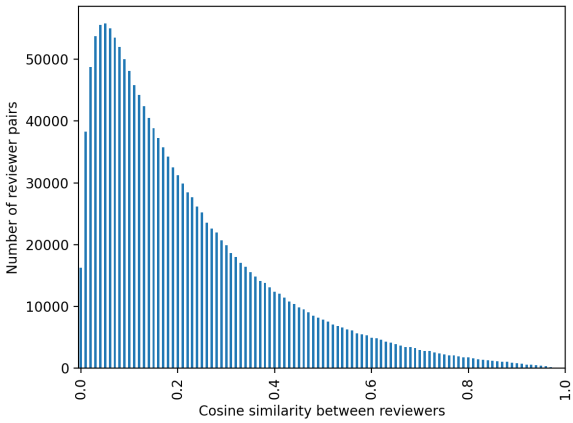
In the previous sections, we saw that an attacker has a good success rate of both selecting or avoiding a single reviewer. However, it is likely that in a real scenario, an attacker might want to select one reviewer while simultaneously avoiding other reviewers. Moreover, it is likely that these reviewers have similar areas of expertise.

We test the ability of our attack to favor one reviewer while simultaneously avoiding another, similar reviewer. For each pair of reviewers, we select one reviewer as the favored reviewer and then create an adversarial submission that maximizes the difference between the ranks of the two reviewers. For each topic in the model, we calculate a score as the difference between the probability of the favored reviewer being an expert in that topic and the probability of the avoided reviewer being an expert in that topic. Thereafter, we select the most likely word from each topic and weight the probability of its occurrence by its topic's score. If a word is most likely for more than one topic, we weight it by the highest scoring topic. We then use this probability distribution over the words to create an adversarial text of 1000 words. We embed this adversarial text in our submission and hide the original text from Autobid.

Fig 7a shows the resultant difference in the ranks of reviewers in a pair against the cosine similarity of the reviewers. To calculate the cosine similarity between reviewers, we represent each reviewer as a vector of that reviewer's expertise probability distribution over all topics. As seen in the figure, our approach works fairly well for less similar reviewers. The rank difference falls below 500 only when the similarity between reviewers is greater than 0.9. To put this



(a) Difference in rank, for an adversarial submission, of reviewers in a pair vs. the cosine similarity of the reviewers.



(b) Distribution of reviewer pairs over the range of cosine similarity.

Fig. 7: Maximizing the difference in rank between two reviewers for an adversarial submission.

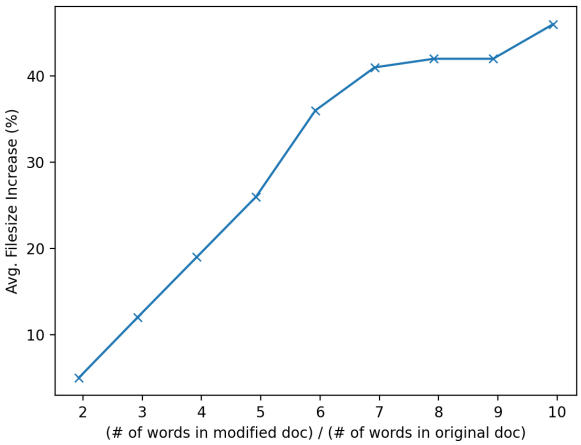


Fig. 8: Average increase in file size as the number of words in the modified document increases.

into perspective, Fig 7b shows the distribution of the reviewer pairs over the range of similarity scores. This suggests that most of the pairs of reviewers have a low similarity score and only 0.28% of the pairs have a similarity score greater than or equal to 0.9. Therefore, for more than 99.7% of reviewer pairs, this strategy leads to a difference of more than 500 in the ranks of the reviewers for the adversarial submission.

E. File Size

We were initially worried that the increase in file size between the original and modified PDFs would be an indicator of our tampering. Fig 8 shows the increase in file size as the number of words injected into the adversarial submission is increased. When the adversarial document contains approximately 10 times the number of words contained in the original document, the file size of the adversarial document increases by an average of 46 percent as compared to the original document. One reason for this low increase is that our method of embedding text currently uses the entire area of a page to embed the adversarial words, whereas the original documents have greater margins, figures and tables. This means that the overlay PDF we use to embed the text does not require as many pages as the original document, which helps keep the overall file size lower. Another reason is that because we are adding repeated words, this allows for greater compression of the adversarial text. It is easier to compress 1000 repeated occurrences of a single word than it is to compress 1000 different words.

IX. Potential Mitigations

In our current implementation, a reviewer could easily identify a modified PDF by simply using their mouse to highlight white text in an area where no text should be. However, with a more sophisticated implementation, we could embed adversarial words only behind the existing text in the unmodified document, making it more difficult for a reader to notice discrepancies in the text selection. We therefore do not see this as a valid mitigation.

As described above, we saw that our rate of success in tricking Autobid was approximately 50 percent when the number of words in the modified document was 3 times the number of words in the original document and the original text was not ignored. At this word ratio, we see a 12 percent average increase in the file size of the modified document. For most submissions, this means the file size of the modified document should fall within any reasonable range for acceptable, unmodified submissions. Hence, mitigations based on file size are unlikely to flag modified submissions without also experiencing many false positives on unmodified submissions. Additionally, hiding the original text from the text extractors can keep the file size small because fewer keywords need to be embedded in the adversarial submission. We therefore do

not see file size analysis as a valid mitigation either.

As previously mentioned, `pdftotext` includes options to exclude text outside the content bounding boxes, text drawn in PDF render mode 3 (invisible), and text with opacity less than 0.001. However, as far as we are aware, it does not include an option to exclude white-colored text. This means that a mitigation based on the color of the text would require changes to `pdftotext` or the use of a different PDF parser that does provide such an option. To prevent a variety of methods of embedding text, the parser should ignore any text below a certain font size, any text without full opacity, and any text that is not colored black or nearly black. These restrictions should prevent many ways of visually hiding embedded text.

In our current approach, we embed a few adversarial keywords multiple times (repeated continuously to drive the word occurrence probability to a favorable value) in the submission. Thus, one other mitigation can be to analyze the parsed text from the submitted PDF for unusually high number of repetitions for a few words. This heuristic may be used to flag some submissions as potentially adversarial which can then be manually verified.

Despite the limitations of our current implementation, we submitted a few modified papers to a system using HotCRP [2], successfully passing the style and layout checks that HotCRP runs on the submissions. These tests were performed manually and would need to be automated to ensure HotCRP will accept all modified submissions, but our initial success is promising.

X. Related Work

Plenty of work on embedding data into PDFs already exists. However, much of this work focuses on using the PDF as a medium for passing secret messages without explicitly writing them, such as [16] or [10]. We were unable to find any work on embedding messages specifically so that they can be extracted by PDF parsers. In this work, we show that it is possible to trick PDF parsers not only into extracting embedded adversarial text but also into ignoring the original text.

More work exists on exploiting the conference review process, such as [13], [15] and [9]. However, these works focus on an adversary who is an author of a submission and also a reviewer in the conference. These works show that the reviewers can submit adversarial bids and reviews, and collude with other reviewers, to impact the bids and reviews for their authored submission. In comparison, we consider an adversary who is an author of a submission but not necessarily a reviewer in the conference and can only manipulate her own submission.

XI. Future Work

As described earlier, our implementation of embedding text is naive to the internal structure of PDF documents. A better embedding process would be to take the format of the original document into account so that adversarial text is only embedded behind existing text. Additionally, the author of the submission can attempt to work the adversarial words into the actual text of the document in natural places and prevent the other text from being extracted. Because the adversarial words are the only words extracted from the document, the author of the submission does not need to include them multiple times, and the desired reviewer should still be highly ranked. This process would subvert all mitigations raised in the previous section. However, such a process could not be done fully automatically, as the author of the submission must manually include the adversarial words.

Currently we choose our adversarial keywords based on the topic and word probabilities generated by the same LDA model that is used to generate the bids. However, as the attackers would not have access to the exact model, it would be more realistic to use an approximation of the true model, perhaps generated using a corpus of only reviewers' publications, in order to choose the adversarial words. Moreover, applying our attack on a system based on a different model (e.g. StarSpace [14]) will be a good test of the generalization of this approach. In our current approach, we add a few adversarial words multiple times which may make it easy to mitigate our attack. One direction of future research can be to obtain successful attacks with an upper limit on the number of times a particular word can be repeated.

We found that some reviewers are more difficult to match with than others. It would be interesting to examine what features of a reviewer's publications influence the ability to select them, as well as whether there are different keyword choices that can improve success with difficult reviewers.

XII. Conclusion

The assignment of papers to reviewers is a difficult problem facing conference organizers. We show that it is possible to modify a paper submission in such a way as to force TPMS, a popular solution to this problem, to pick or avoid arbitrary reviewers for the paper. Specifically, we embed keywords into the PDF so that there are no visible changes to the document but TPMS determines the paper is related to the desired reviewer's area of expertise and unrelated to the unwanted reviewer's area of expertise. We show the effectiveness of our attack through various experiments and find that an attacker can get her paper reviewed by a favorable reviewer with a probability as high as 80% and can avoid an unfavorable reviewer with a probability as high as 99%. The implementation of the attack described in this paper is mostly naive to the internal structure of PDF documents and could therefore be defeated by a few simple mitigations. However, more complex attacks based on the same principles but taking the internals of the PDF format into account could be difficult to mitigate.

[2] https://test.hotcrp.com/

References

[1] *NeurIPS 2019*, 2019 (accessed Mar 11, 2020). [Online]. Available: https://nips.cc/Conferences/2019/Reviewers

[2] *NeurIPS 2019*, 2019 (accessed Mar 11, 2020). [Online]. Available: https://nips.cc/Conferences/2019

[3] *NeurIPS Proceedings*, 2019 (accessed Mar 11, 2020). [Online]. Available: https://papers.nips.cc/

[4] *PyPDF4*, 2020 (accessed Apr 13, 2020). [Online]. Available: https://github.com/claird/PyPDF4

[5] *ReportLab*, 2020 (accessed Apr 13, 2020). [Online]. Available: https://www.reportlab.com/opensource/

[6] *pdftotext*, 2020 (accessed Mar 18, 2020). [Online]. Available: https://www.xpdfreader.com/pdftotext-man.html

[7] L. Charlin and R. Zemel, "The toronto paper matching system: an automated paper-reviewer assignment system," 2013.

[8] D. Charrez, *NeurIPS 2019 Stats*, 2019 (accessed February 17, 2020). [Online]. Available: https://medium.com/@dcharrezt/neurips-2019-stats-c91346d31c8f

[9] A. Kahng, Y. Kotturi, C. Kulkarni, D. Kurokawa, and A. Procaccia, "Ranking wily people who rank each other," 2018. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17019/15796

[10] I.-S. Lee and W.-H. Tsai, "A new approach to covert communication via pdf files," 2010.

[11] B. Parno, *Autobid*, 2018 (accessed Mar 11, 2020). [Online]. Available: https://github.com/parno/autobid/tree/master

[12] D. Stevens. (2019) Pdf tools. [Online]. Available: https://blog.didierstevens.com/programs/pdf-tools/

[13] S. Thurner and R. Hanel, "Peer-review in a world with rational scientists: Toward selection of the average," *The European Physical Journal B*, vol. 84, pp. 707–711, 2010.

[14] L. Wu, A. Fisch, S. Chopra, K. Adams, A. Bordes, and J. Weston, "Starspace: Embed all the things!" 2017.

[15] Y. Xu, H. Zhao, X. Shi, and N. B. Shah, "On strategyproof conference peer review," *CoRR*, vol. abs/1806.06266, 2018. [Online]. Available: http://arxiv.org/abs/1806.06266

[16] S. Zhong, X. Cheng, and T. Chen, "Data hiding in a kind of pdf texts for secret communication," 2007.