IT458 - Information Retrieval
Document Content Extraction for Slide Generation

Bhagyashri Bhamare(181IT111)
C Sneha(181IT112)
Karthik R(181IT235)

Agenda

- Introduction
- Literature Survey
  - Outcome of Literature Survey
- Problem Statement
- Objectives
- Proposed Methodology
- Results
- Individual Contribution
- References

Introduction

- From business to education to research, presentations are everywhere as they are visually effective in summarizing and explaining bodies of work to the audience
- Presentations are critical for communication in all areas of our lives, yet the creation of slide decks is often tedious and time-consuming.
- Automation in the process of creating slides when a document with the information exists would save a lot of precious time and effort.

Literature Review

Authors	Methodology	Advantages	Limitations
“Document to slide generation via query based text summarization.” Edward Sun, Yufang Hou, Dakuo Wang, Yunfeng Zhang, Nancy X.R. Wang	1) Use slide titles to retrieve relevant and engaging text, figures, and tables 2) Summarize the retrieved context into bullet points with long-form question answering	Uses abstractive summarization instead of extraction based method to summarize important points. Can create multiple slides under same title instead of one-to-one match	Lack of ability to handle long text sequence
“Automatic slide generation for scientific papers.” Ather Seifid, Jian Wu, Lee Giles	Extractive summarization techniques are applied to rank and select important sentences.	Works efficiently as it’s extraction based summarization. Less time to train the model.	Extraction based method will have a trade off for conjunction of sentences in an appealing format.

Outcome of Literature Review

The existing research works have yielded promising progress towards the goal of automated slide generation, but they also face two common limitations:

1) Most works primarily rely on extractive-based mechanisms which are , thus the generated content is merely an aggregation of raw sentences from the document, whereas in real-world slides, the presenter frequently uses abstractive summarization

2) Moreover it gives a paragraph of of content whereas presenters generally prefer the presentation to have a hierarchical structure with bullet points.

Problem Statement

To implement a document content extraction for generating slide.

Research Objectives

1. Data Pre-processing(converting pdfs to text documents.)
2. Extracting information from document that is relevant.
3. Effective summarizing of research paper.
4. Display data in an appealing format in slide.
5. Evaluation of slide content generated.

Methodology

Dataset -

- we are using slide-paper dataset consisting of 5,000 scientific articles and corresponding manually made slides.
- Training: 4500 examples.
- Test: 500 examples.

Preprocessing Data

- Extraction of the pdf documents and slides in .xml format
- Extraction of the slide documents in .xml format.

Methodology

Sentence Labeling:

- Generate extractive labels for sentences of the input document
- Sentence labeling process attempts to identify salient sentences that are semantically similar to the corresponding slides.
- Generates an extractive summary, which will be used as the ground truth for training and evaluation
- SummaRuNNer treats the summarization task as a sequence labeling problem, if adding the sentence to the summary improves the ROUGE score, the sentence is labeled with 1, otherwise it is labeled with 0

Methodology continued....

Model Training
We are using Bidirectional long short-memory term (BiLSTM) to get a probability score of each sentence which is important based on the following:

- Position
- Novelty
- Content

score(sentence) = σ(pos + content + novelty)

Methodology continued....

Loss is calculating by

```
loss = target * log(output+c) + (1-target) * log(1 - output+c)
```

USB provides ubiquitous plug-and-play connectivity for a wide range of devices.
However, the complex nature of USB obscures the true functionality of devices from the user, and operating systems blindly trust any physically-attached device.
This has led to a number of attacks, ranging from hidden keyboards to network adapters, that rely on the user being unable to identify all of the functions attached to the host.
In this paper, we present USBFILTER, which provides the first packet-level access control for USB and can prevent unauthorized interfaces from successfully connecting to the host operating system.
USBFILTER can trace individual USB packets back to their respective processes and block unauthorized access to any device.
By instrumenting the host's USB stack between the device drivers and the USB controller , our system is able to filter packets at a granular-ity that previous works cannot-at the lowest possible level in the operating system.
USBFILTER is not only able to block or permit specific device interfaces; it can also restrict interfaces to a particular application (e.g., only Skype can access my webcam).
Furthermore, our experimental analysis shows that USBFILTER introduces a negligible (3-10µs) increase in latency while providing mediation of all USB packets on the host.
Our system provides a level of granularity and extensibility that reduces the uncertainty of USB connectivity and ensures unauthorized devices are unable to communicate with the host.
The Universal Serial Bus (USB) provides an easy-to-use, hot-pluggable architecture for attaching external devices ranging from cameras to network interfaces to a single host computer.
USB ports are pervasive; they can often be found on the front, back, and inside of a common desktop PC.
Furthermore, a single USB connector may connect multiple device classes.
These composite devices allow disparate hardware functions such as a microphone and speakers to appear on the same physical connector (e.g., as provided by a headset).
In the host operating system, technologies such as USBIP [21] provide the capability to remotely connect USB devices to a host over a network.
The result is a complex combination of devices and functionalities that clouds the user's ability to reason about what is actually connected to the host.Attacks that exploit this uncertainty have become more prevalent.
Firmware attacks such as BadUSB [27] modify benign devices to have malicious behavior (e.g., adding keyboard emulation to a storage device or perform automatic tethering to another network).
Hardware attacks [1] may inject malware into a host, provide RF remote control capabilities, or include embedded proxy hardware to inject and modify USB packets.
Attackers may also exfiltrate data from the host by leveraging raw I/O (e.g., using libusb [14]) to communicate with the USB device directly, or bypass the security mechanism employed by the USB device controller by sending specific USB packets to the device from the host USB controller [4].
Unfortunately, the USB Implementers Forum considers defending against malicious devices to be the responsibility of the user [44], who is unlikely to be able to independently verify the functionality and intent of every device simply by its external appearance, and may just plug in USB devices to take a look [43].
Modern operating systems abstract USB authorization to physical control, automatically authorizing devices connected to the host, installing and activating drivers, and enabling functionality.
We believe that a finergrained control over USB is required to protect users.
In this paper, we make the following contributions:• Design and develop a fine-grained USB access control system: We introduce USBFILTER, a packet-level firewall for USB.
Our system is the first to trace individual USB packets back to the source or destination process and interface.
USBFILTER rules can stop attacks on hosts by identifying and dropping unwanted USB packets before they reach their destination in the host operating system.
• Implement and characterize performance: We demonstrate how USBFILTER imposes minimal overhead on USB traffic.
As a result, our system is well-suited for protecting any USB workload.
• Demonstrate effectiveness in real-world scenarios: We explore how USBFILTER can be used to thwart attacks and provide security guarantees for benign devices.
USBFILTER can pin devices (e.g., webcams) to approved programs (e.g., Skype, Hangouts) to prevent malicious software on a host from enabling or accessing protected devices.USBFILTER is different from previous works in this space because it enables the creation of rules that explicitly allow or deny functionality based on a wide range of features.
GoodUSB [41] relies on the user to explicitly allow or deny specific functionality based on what the device reports, but cannot enforce that the behavior of a device matches what it reports.
SELinux [35] policies and PinUP [13] provide mechanisms for pinning processes to filesystem objects, but USBFILTER expands this by allowing individual USB packets to be

Fig-1 Document in .txt format

Fig-2 Label for each sentence

Fig 3 Predicted scores of the sentences

Fig4 sentences in human generated slide

By instrumenting the host's USB stack between the device drivers and the USB controller , our system is able to filter packets at a granular-ity that previous works cannot-at the lowest possible level in the operating system. Attackers may also exfiltrate data from the host by leveraging raw I/O (e.g., using libusb [14]) to communicate with the USB device directly, or bypass the security mechanism employed by the USB device controller by sending specific USB packets to the device from the host USB controller [4]. USBFILTER can pin devices (e.g., webcams) to approved programs (e.g., Skype, Hangouts) to prevent malicious software on a host from enabling or accessing protected devices.USBFILTER is different from previous works in this space because it enables the creation of rules that explicitly allow or deny functionality based on a wide range of features. The granularity and extensibility of USBFILTER allows it to perform the functions of existing filters [41] while permitting much stronger control over USB devices.The remainder of this paper is structured as follows: In Section 2, we provide background on the USB protocol and explain why it is not great anymore; in Section 3, we discuss the security goals, design and implementation of our system; in Section 4, we discuss how USBFILTER meets our required security guarantees; in Section 5, we evaluate USBFILTER and discuss individual use cases; in Section 6, we provide additional discussion; in Section 7, we explore related work; and in Section 8, we conclude. A USB device refers to a USB transceiver, USB hub, host controller, or peripheral device such as a human-interface USB Device Interface 0 Interface 1 Interface 2 In Out In In Out Out EP 0 EP 0 EP 1 EP1 EP 0 EP 0 EP 0 EP 0 EP 0 EP 1 EP 1 EP 1 EP 1 EP 2 EP 2 EP 2 EP 2 EP n EP n Figure 1: A detailed view of a generic USB device. The USB protocol works in a master-slave fashion, where the host USB controller is responsible to poll the device both for requests and responses. When a USB device is attached to a host machine, the host USB controller queries the device to obtain the configurations of the device, and activates a single configuration supported by the device. By parsing the current active configuration, the host operating system identifies all the interfaces contained in the configuration, and loads the corresponding device drivers for each interface. The relative ease with which a USB peripheral can be installed on a host is simultaneously its greatest and most insecure property.The USB subsystem has been expanded in software as well, with Virtio [30] supporting I/O virtualization in KVM, enabling virtual USB devices in VMs, and passing through the physical devices into VMs. Wireless USB (WUSB) [19] and Media Agnostic USB (MAUSB) [16] promote the availability of USB devices by leveraging different wireless communication protocols, making the distinction among local USB devices, virtual ones, and remote ones vanish.Overall, the utility and complexity of USB has been steadily increasing in both hardware and software. Attacks such as BadUSB [27] and TURNIPSCHOOL [1] (itself designed on specifications from nation-state actors) use composite devices to present multiple interfaces to a host. Since operating systems implicitly trust any device attached, these hidden functions are enumerated, their drivers are loaded, and they are granted access to the host with no further impediment.Data exfiltration from host machines may be the main reason why USB storage is banned or restricted in enterprise and government environments. While access controls can be bypassed by raw I/O, which communicates to the device directly from userspace (e.g., using libusb [14]), network-based methods are vulnerable to network spoofing (e.g., ARP spoofing [32] and DNS spoofing [36]). The adversary may also launch network attacks in order Figure 2: USBFILTER implements a USB-layer reference monitor within the kernel, by filtering USB packets to different USB devices to control the communications between applications and devices based on rules configured.to enable or access authorized devices from unauthorized processes or devices. • Unauthorized Access: The adversary may attempt to enable or access authorized devices on a host (e.g., webcam, microphone, etc.) via unauthorized software to gain access to information or functionality that would otherwise inaccessible.We assume that as a kernel component, the integrity of USBFILTER depends on the integrity of the operating system and the host hardware (except USB devices). The core USBFILTER component is statically compiled and linked into the Linux kernel image, which hooks the flow of USB packets before they reach the USB host controller which serves the USB device drivers, as shown in Figure 3. Furthermore, applications generally submit asynchronous I/O requests, causing the kernel to perform the communications task on a separate background thread.This problem also appears when inspecting USB network device packets, including both wireline (e.g., Ethernet) dongles and wireless (e.g., WiFi) adapters. In these cases, USB is an intermediate layer to encapsulate IP packets into USB packets for processing by the USB networking hardware.These cases are problematic for USBFILTER because a naïve traceback approach will often only identify the kernel thread as the origin of a USB packet. Once verified, new rules will be synchronized with the kernel and saved locally.If no user-defined rules are present, USBFILTER enforces default rules that are designed to prevent impact on normal kernel activities (e.g., USB hot-plugs). The TPM provides a root of trust that allows for a measured boot of the system and provides the basis for remote attestations to prove that the host machine is in a known hardware and software configuration. Devices cannot initiate USB packet transmission without permission from the controller.We also instrument the virtual USB host controller (vhci) to cover virtual USB devices (e.g., USB/IP). Future work will add general rules, which can be overwritten by new rules.-d|--debug enable debug mode -c|--config path to configuration file (TBD) -h|--help display this help message -p|--dump dump all the rules -a|--add add a new rule -r|--remove remove an existing rule -s|--sync synchronize rules with kernel -e|--enable enable usbfilter -q|--disable disable usbfilter -b|--behave change the default behavior -o| --proc process table rule -v|--dev device table rule -k|--pkt packet table rule -l|--lum LUM table rule -t|--act table rule action ----------proc: pid,ppid,pgid,uid,euid,gid,egid,comm dev: busnum,devnum,portnum,ifnum,devpath,product, manufacturer,serial pkt: types,direction,endpoint,address lum: name behavior/action: allow|drop Figure 4. In this setting, malicious devices have to impersonate benign devices to allow communications, which are still regulated by the rules, e.g., no HID traffic allowed for a legit USB storage device. We can write USBFILTER rules to completely shutdown the keyboard and mouse functionalities:usbtables -a teensy1 -v ifnum=2,manufacturer= Teensyduino,serial=1509380 -t drop usbtables -a teensy2 -v ifnum=3,manufacturer= Teensyduino,serial=1509380 -t dropIn these rules, we use condition "manufacturer" and "serial" (serial number) to limit the Teensy's functionality. Then we add following rules into the host machine:usbtables -a mymouse -v busnum=1,devnum=4,portnum=2, devpath=1.2,product="USB Optical Mouse", manufacturer=PixArt -k types=1 -t allow usbtables -a mykeyboard -v busnum=1,devnum=3, portnum=1,devpath=1.1, product="Dell USB Entry Keyboard", manufacturer=DELL -k types=1 -t allow usbtables -a noducky -k types=1 -t dropThe first two rules whitelist the existing keyboard and mouse on the host machine; the last rule drops any USB packets from other HID devices. However, we have leveraged information about the physical interface (busnum and portnum) to write the first two rules, which would require the attacker to unplug the existing devices, plug the malicious device in, and impersonate the original devices including the device's VID/PID and serial number. In this case study, we use a Kingston DT 101 II 2G USB flash drive, and insert the following rule:usbtables -a nodataexfil -v manufacturer=Kingston -l name=block_scsi_write -t dropThis rule prevents modification of files on the storage device. We can also block any writes to USB storage devices:usbtables -a nodataexfil4 -l name=block_scsi_write -t drop USBFILTER logs dropped USB packets, and these logs can easily be used in a centralized alerting system, notifying administrators to unauthorized access attempts. In this case study, we show how to use USBFILTER to restrict the use of a Logitech Webcam C310 to specific users and applications.usbtables -a skype -o uid=1001,comm=skype -v serial=B4482A20 -t allow usbtables -a nowebcam -v serial=B4482A20 -t dropThe serial number of the Logitech webcam is specified in the rules to differentiate any others that may be attached to the system as well as to prevent other webcams from being attached. The phone is unable to be used for storage or tethering after the rule is applied.We can construct a more specific charge-only rule:usbtables -a charger -v busnum=1,portnum=4 -t dropThis rule specifies a specific physical port on the host and this port can only be used for charging. The overhead of USB enumeration introduced by

Fig 5 System generated slide

ROUGE

- Recall-Oriented Understudy for Gisting Evaluation.
- Set of metrics for evaluating automatic summarization of texts.
- It works by comparing an automatically produced summary against a set of reference summaries (human-produced).

Different Types:

- Precision
- Recall
- F score

Recall

How much of the reference summary the system summary is recovering.

Counts the number of overlapping n-grams found in both the model output and reference

Divides this number by the total number of n-grams in the reference.

Calculated as follows:

$$\frac{\text{count}_{\text{match}}(\text{gram}_n)}{\text{count}(\text{gram}_n)}$$

Precision

Counts the number of overlapping n-grams found in both the model output and reference

Divides this number by the total number of n-grams in the reference.

Calculated as follows:

$$\frac{\text{count}_{\text{match}}(\text{gram}_n)}{\text{count}(\text{gram}_n)}$$

F1 Score

Weighted average of Precision and Recall

Calculated as:

$$2 * \frac{precision * recall}{precision + recall}$$

Results

SIze	Recall	Precision	F
R-1	0.63	0.80	0.67
R-2	0.47	0.60	0.50
R-3	0.41	0.54	0.45
R-4	0.39	0.51	0.43

```
1 ROUGE-1 Average_R: 0.63840 (95%-conf.int. 0.61889 - 0.65706)
1 ROUGE-1 Average_P: 0.80401 (95%-conf.int. 0.77939 - 0.82680)
1 ROUGE-1 Average_F: 0.67875 (95%-conf.int. 0.65954 - 0.69593)
---------------------------------------------
1 ROUGE-2 Average_R: 0.47052 (95%-conf.int. 0.45523 - 0.48599)
1 ROUGE-2 Average_P: 0.60800 (95%-conf.int. 0.58525 - 0.62995)
1 ROUGE-2 Average_F: 0.50857 (95%-conf.int. 0.49187 - 0.52478)
---------------------------------------------
1 ROUGE-3 Average_R: 0.41460 (95%-conf.int. 0.39922 - 0.42932)
1 ROUGE-3 Average_P: 0.54028 (95%-conf.int. 0.51845 - 0.56224)
1 ROUGE-3 Average_F: 0.45068 (95%-conf.int. 0.43365 - 0.46718)
---------------------------------------------
1 ROUGE-4 Average_R: 0.39753 (95%-conf.int. 0.38173 - 0.41226)
1 ROUGE-4 Average_P: 0.51876 (95%-conf.int. 0.49689 - 0.54048)
1 ROUGE-4 Average_F: 0.43244 (95%-conf.int. 0.41535 - 0.44869)
---------------------------------------------
1 ROUGE-L Average_R: 0.44967 (95%-conf.int. 0.43378 - 0.46505)
1 ROUGE-L Average_P: 0.58156 (95%-conf.int. 0.55654 - 0.60416)
1 ROUGE-L Average_F: 0.48486 (95%-conf.int. 0.46693 - 0.50146)
---------------------------------------------
1 ROUGE-W-1.2 Average_R: 0.06704 (95%-conf.int. 0.06305 - 0.07103)
1 ROUGE-W-1.2 Average_P: 0.37671 (95%-conf.int. 0.35868 - 0.39329)
1 ROUGE-W-1.2 Average_F: 0.10760 (95%-conf.int. 0.10212 - 0.11299)
---------------------------------------------
1 ROUGE-S* Average_R: 0.42924 (95%-conf.int. 0.40846 - 0.44950)
1 ROUGE-S* Average_P: 0.68277 (95%-conf.int. 0.65316 - 0.71003)
1 ROUGE-S* Average_F: 0.46626 (95%-conf.int. 0.44562 - 0.48503)
---------------------------------------------
1 ROUGE-SU* Average_R: 0.42956 (95%-conf.int. 0.40883 - 0.44982)
1 ROUGE-SU* Average_P: 0.68296 (95%-conf.int. 0.65337 - 0.71019)
1 ROUGE-SU* Average_F: 0.46657 (95%-conf.int. 0.44596 - 0.48528)
```

Conclusion

In our work we have presented our methodology for extracting text and summarisizng to generate the content for slides given a research or academic paper using SummaRuNNer.
In the future we look to explore other models and expand our dataset to other languages such as Chinese and Russian as most academic works are published in these languages and also include extracting images and tables.

References

Base Paper

Edward Sun, Yufang Hou, Dakuo Wang, Yufeng Zhang, Nancy X.R. Wang, 'Document to slide generation using query based text-summarization' https://arxiv.org/pdf/2105.03664v1.pdf

- K. Gokul Prasad, H. Mathivanan, T. V. Greetha and M. Jayaprakasam, "Document Summarization and Information Extraction for Generation of Presentation Slides," *2009 International Conference on Advances in Recent Technologies in Communication and Computing*, 2009, pp. 126-128, doi: 10.1109/ARTCom.2009.74
- Ather Seifid, Jian Wu, Lee Giles "Automatic slide generation for scientific papers." SciKnow '19, November 19–22, 2019, Marina del Rey, CA
- Wu, Tiantian & Yu, Hongzhi & Wan, Fucheng & Yang, Fangtao. (2019). Research on Answer Extraction for Automatic Question Answering System. 10.2991/iccia-19.2019.34.
- Sharma, Lokesh & Mittal, Namita. (2018). Answer Extraction in Question Answering using Structure Features and Dependency Principles.
- Raju Barskar, Gulfishan Firdose Ahmed, Nepal Barskar, ,'An Approach for Extracting Exact Answers to Question Answering (QA) System for English Sentences',Procedia Engineering, Volume 30, 2012, ISSN 1877-7058,

Thank you