A web-based TEX previewer: the ecstasy and the agony

Michael Doob
Department of Mathematics
The University of Manitoba
Winnipeg, Manitoba, Canada R3N 2T2
mdoob@ccu.umanitoba.ca

Abstract

The appeal of a web-based combined TEX editor and previewer is instantaneous. It allows not only the easy testing of snippets of code, the writing of short abstracts and even of short papers, but also allows sharing of the results over the web. Unfortunately, even a benign program like TEX presents serious security risks, and care must be used when exposing such an application.

This presentation includes a web-based viewer of the type just described. It will be used to:

- Illustrate how remarkably easy it is, using tools readily available, to construct a previewer,
- give examples of potential security problems, and
- indicate some solutions to these problems.

The context of this talk is a LAMP (Linux, Apache, MySQL, PHP) environment, but the basic ideas can be applied to any of the common operating systems.

1 Ecstasy

The appeal of a web-based TEX previewer is immediate. There are many possible reasons for this. We start with some of the them.

1.1 Motivation

1.1.1 Remote Access

We at the Publications Office of the Canadian Mathematical Society receive papers accepted for publication (sometimes called a sow's ear) in many different levels of quality of TEX. They must all be made to conform to our publication standards (sometimes called a silk purse), and significant manpower is used for this purpose. We have a number of editors who work both at our office and at home. There is no problem putting TEX on a home computer. We have our own style file, and that can be put on the home computers too (although it does change from time to time). However, there is a significant problem with our fonts. We have a number of proprietary (Adobe) fonts, and the license restricts their distribution. The TFM files are no problem and can be put on the home computers; the only problem is with the previewing since that uses the proprietary information. Hence a web page previewer with a one-button upload of the TEX file followed by running LATEX with our class file and then displaying the resulting pages is just what we need.

1.1.2 Abstract Submissions

The Canadian Mathematical Society has semiannual meetings in June and December. There are several hundred abstracts for each meeting which need to be in LATEX format compatible with the style of our proceedings. Our traditional method was to allow presenters to submit their (purported) LATEX files by email. Changing these sows' ears into silk purses uses significant resources. With a web page the author can edit the LATEX file until it works properly with our style file.

We now provide a window into which the abstract may be loaded. It can be run though the appropriate version of LATEX and, if needed, can be further edited and rerun within the same window. This transfers the editing efforts from our personnel to the author. There is, of course, a resulting decrease in quality due to author inability to use LATEX optimally. The abstracts are ephemeral (they are used for the one meeting only), and so this is an acceptable cost.

1.1.3 Snippet Testing

Sometime it's desirable to try out a new definition that may take a few tries to get it right. If the web

server is on a local machine, the turnaround time is instantaneous. It's easy to incrementally improve the code until it is perfect.

Similarly, it is useful to use the picture environment incrementally to create figures that will be usable with any implementation of LaTeX.

If you subscribe to `texhax <texhax@tug.org>`, then lots of little problems that arise from that list can be checked and/or debugged on the spot.

1.1.4 Because We Can

The improvements in the speed of software applications used with web browsers over the past few years have been breathtaking. We have long been able to run TeX on a local machine and view the output immediately on a previewer. It is interesting that we can replicate that experience using a reasonable web connection.

1.2 LAMP Implementation

1.2.1 Environment

Our environment used for this application is sometimes called LAMP: the Linux operating system, the Apache web server, the MySQL database management system (unused in this application) and PHP (sometimes the "P" is Perl or Python; indeed, either could be used instead of PHP). No extra modules are used with Apache, and no additional packages are loaded into PHP.

1.2.2 Desired Elements

The minimum implementation would allow input (an input window using direct typing, cut-and-paste or file upload) as well as output that is dependent on the success or failure of the TeX job. It's also easy to have only file uploads and to display (portions of) the log file.

Additionally, it's also possible to preload TeX input or specific packages. For example, it could be more convenient to have the material in the input window inserted between the lines:

```
\documentclass{article}
\begin{document}

\end{document}
```

Similarly, it's also easy to preload either document classes of packages using pulldown menus. Examples are given in the documentation.

1.2.3 Browser Peculiarities

Ideally simple output should be rendered identically by different browsers. This ideal, unfortunately, is not met. For example, the output from rerunning TeX should reflect the content in the current input window. In fact, there is an html metacommand for exactly this purpose:

```
<META HTTP-EQUIV="CACHE-CONTROL"
      CONTENT="NO-CACHE">
```

Alas, some browsers will ignore this command, but these shortcomings can be overcome in a LAMP environment. It's always possible to generate unique names with each call to TeXto avoid the cache problem. It's also possible to use freely available software to generate graphics (`png`, `jpg`, `pdf` or `svg`) whose renderings will be (more or less) browser independent.

2 Agony

As can be seen in the accompanying documentation, it's easy to set up a web-based TeX previewer within a LAMP environment. Alas, as with any web application that may be accessed widely, there are certain concerns and possible exploits that must be addressed. At first blush, TeX is pretty robust and locks out the most dangerous threats. For example, there are no system calls available. Nonetheless, there are precautions that must be taken.

2.1 Need to know

Clearly, the more widespread the audience is for a web application, the less is the information that should be disclosed about the the operating environment. There are two options: control the access to the web pages or control the amount of information disclosed. In a LAMP environment this is easy.

It is a standard configuration command for the Apache server to restrict access to some (or even all) directories to clients with specific internet addresses, so the access, if desired, may be localized.

On the other hand, the log file, even when there is only one line of input, will reveal information about the operating system:

```
This is TeX, Version 3.14159 (Web2C 7.4.5)
/usr/share/texmf/tex/latex/base/size10.clo
```

Loading more packages and fonts generates similar messages concerning the versions running and the structure of the file system. These may and should be filtered out when the log file is requested. This same is true for error messages.

2.2 Denial of Service

Denial of Service (DOS) attacks are designed to utilize all of the resources available on a particular computer and thus deny access by others. There are several methods by which this may be done.

2.2.1 CPU hogging

Consider what happens with the following LATEX input:

```
\newcounter{cnt}
\loop
   \thecnt\newpage \stepcounter{cnt}
   \ifnum \value{cnt}<10000
\repeat
```

This produces a 10,000 page document with one integer (actually two if you include the page number) on each page. Suppose the `\stepcounter{cnt}` is left out. Then the loop is infinite, and TEX happily runs until it reaches its memory limit and then halts. Now suppose that `\thecnt\newpage` is also omitted. Then no memory is used, and TEX will run indefinitely using up any cpu resources available. There are two solutions for this:

- Any standard implementations of Linux comes with the `pam` (pluggable authentication module) software. This module uses a file called `limits.conf` to control, among other things, the amount of cpu time any process can use.
- For operating systems without `pam` there is a program called `cpulimit` which may be used to control the percentage of available cpu resources that may be allocated to a given process.

2.2.2 Disk hogging

Now consider the following LATEX input:

```
\newcounter{cnt}
\loop
   \leavevmode\newpage \stepcounter{cnt}
   \ifnum \value{cnt}<10000
\repeat
```

This produces a 10,000 page document with only the page numbers on each page (of course, the use of `\pagestyle{empty}` will make the page completely blank). If we delete the `\stepcounter{cnt}` from the input, then TEX runs indefinitely using no memory, but the dvi file will (apparently) grow without limit.

This problem is easy to address. The file mentioned above, `limits.conf`, can also control disk usage. Alternatively, disk quotas, turned off by default, may be enabled.

2.2.3 Server hogging

Any web application is subject to attack through the server. A distributed DOS attack, that is, one from a botnet of clients is really impossible to stop. Even with web pages, the mouse clicks can be spoofed, so it is important to keep the web applications isolated from the rest of the computer environment.

2.3 Isolation

Putting any application on the web, as we have seen, has inherent dangers. While these can not be eliminated, they can be somewhat mitigated by isolating the web application, inasmuch as possible, from the rest of the computer environment. There are three possible approaches.

2.3.1 Chroot Jail

The `chroot` command is available on all UNIX implementations. All the software (binaries and libraries) needed for the application are put on one directory, and the `chroot` command then limits the operating system access to that directory (and its subdirectories) only. We say that the operating system is in chroot jail. This makes the rest of the computer environment safe even if the application is broken.

2.3.2 Software isolation of the Operating System

It is now fairly easy to set up virtual computers within a UNIX environment. It's possible to take a snapshot of the original setup, and then refresh the installation regularly. This means that any damage can be instantly repaired.

2.3.3 Hardware isolation of the Operating System

The most extreme measure is to put the application on its own platform. This is in effect running the web application as an embedded device. Since a web browser can be run headless, the costs are actually quite modest. It is possible, for example, to set up a mini-ITX board with an enclose, RAM and storage for less than $200.

3 Documentation

Finally, we want the actual PHP code that implements the web-based previewer. This is included in the attached TEX file. Running the file through LATEX prints the documentation along with instructions for extracting the PHP code.