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

Michael Doob

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 article describes a web-based viewer of this type. We will:

- 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 a few 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) at 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 consumes 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

Sometimes it's desirable to try out a new definition that may take a few tries to get 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` (`http://lists.tug.org/texhax`) 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 now replicate that experience using even a modest web connection.

1.2 Some nice implementations

There are already several web-based TEX previewers available. Here are some particularly nice examples:

- Troy Henderson's `http://www.tlhiv.org/ltxpreview`
- Jan Přichystal's `http://tex.mendelu.cz/en`
- Jonathan Fine's `http://www.mathtran.org/toys/jfine/editor2.html`

All of these have the same general pattern: A window for typing input, some method of output display, and options that may be chosen using radio buttons or pulldown menus.

1.3 LAMP Implementation

1.3.1 Environment

Our environment used for this application is sometimes called LAMP: the Linux kernel for the operating system, the Apache web server, the MySQL database management system (unused in this application) and the PHP scripting language (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. In addition, no JavaScript is used.

1.3.2 Desired elements

The minimum implementation would usually display 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. In addition, it's also easy to have file uploads only and to display (portions of) the log file.

It's also possible to preload TEX input or specific packages. For example, it might be more convenient to have the material in the input window automatically inserted within:

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

\end{document}
```

if all of the TEX files will be using `article.cls` and no other packages are needed. Similarly, it's also easy to preload either document classes or packages using pulldown menus. Examples are given in the documentation.

1.3.3 Browser peculiarities

Ideally the 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, but these shortcomings can be overcome in a LAMP environment. It's always possible to generate unique names with each call to TEX to avoid the cache problem. It's also possible to use freely available software to generate output (`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 direct system calls available. Nonetheless, there are precautions that must be taken. Examples follow to illustrate these problems, roughly in increasing order of vulnerability.

2.1 The need to know principle

Clearly, the more widespread the audience is for a web application, the less is the information that should be disclosed about the operating environment. There are two options: control the access to the web pages to reduce the risk or control the amount of information disclosed. In a LAMP environment both are 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. Greater restriction of access may (or may not!) reduce the risk of system compromise.

On the other hand, if there is widespread access, then the log file, even when there is only one line of TEX 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
```

In this case, the structure of the file system is revealed; it has files in a position (under `/usr/share`) that indicates an installation via a package manager on a Unix system rather than a `texlive` installation or some other operating system, and as such it gives hints to the location of the vulnerabilities that any operating environment possesses. 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

A more serious problem is that of Denial of Service (DoS) attacks. These 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
   \stepcounter{cnt}
   \ifnum \value{cnt}<500000
\repeat
```

There could hardly be a simpler loop construct. Running it will do nothing but increment the counter from 0 to 500000 and then quit. This takes a few seconds. If you use a utility (like `top`) to check CPU usage while this is running, you will find it maxed out. If the `\stepcounter{cnt}` is deleted, TeX will run indefinitely, eating up all available CPU resources. As a further insult, the PHP call will freeze the browser, so no termination is possible, even if the program were run by innocent error. Ouch!

Here is another example:

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

This produces a 10,000 page document with one integer on each page (actually two if you include the page number). 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. This indicates the following: as long as the loop is doing anything that uses memory there will be a graceful failure in an accidental infinite loop.

What can be done to keep infinite loops from eating up inappropriate resources? There are at least two remedies 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 many 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. One possibility is to have users register and log into the environment that runs the web-based browser software.

2.3 PHP attacks

In a recent paper [1], Stephen Checkoway, Hovav Shacham, and Eric Rescorla have pointed out a significant vulnerability in the writing and subsequent rereading of PHP scripts. Consider the following code:

```
\newwrite\bummerfile
\openout\bummerfile=badfile.php
\write\bummerfile{<?php}
\write\bummerfile{echo passthru("date");}
\write\bummerfile{phpinfo();}
\write\bummerfile{echo
            passthru("cat /etc/passwd");}
\write\bummerfile{?>}
```

This opens a file called `badfile.php` in the same directory where the DVI file is written and writes in it five lines of PHP code. These implement three commands: a listing of the current time (a typical system call), a listing of all the PHP parameters on the system (a clear violation of need-to-know), and finally a listing of the password file. It should be noted that the subdirectory from where the reading of files is done by the Apache server is easily obtained from the listing of the page source. It typically will have something like `<img src="jail/temp128053010.png">` within it, indicating in this case that the subdirectory being read is called `jail`. Thus by adding `jail/badfile.php` to the original `http` address, the file `badfile.php` is executed.

This vulnerability may be addressed in several ways: the directory from which the image files are

read can be separated from the one where the TEX program writes. Alternatively, if a directory contains an `.htaccess` file which in turn contains a line `php_flag engine off`, then PHP files will not be run from that directory (note that this feature is disabled by default and must be enabled in the Apache configuration file).

2.4 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 several possible approaches.

2.4.1 Single computer

The Apache server has a configuration file that is read when the server starts (often called `httpd.conf`). It allows the server to start at different locations in the file system depending on the calling IP address. In particular the address `127.0.0.1` (also known as `localhost`) is always reserved for the local computer. Setting up a virtual host for that address can ensure that the files are not accessible from any outside address. If you need a web-based TEX previewer to be used by many people on one computer, this is a safe method of implementation.

2.4.2 Small sets of users

The configuration file for the Apache server can also be used to restrict the server to predefined IP addresses. Alternatively, pages can be password protected.

2.4.3 Chroot jail

The `chroot` command is available on all Unix implementations. Copies of all the software (binaries and libraries) needed for the application are put in 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 a chroot jail. This makes the rest of the computer environment safe even if the application is broken.

Running the Apache server in a chroot jail will protect the rest of the operating system. In fact a script may be set up to create the jail automatically. If the software in the jail seems questionable, a new copy can be reconstructed.

2.4.4 Software isolation of the operating system

An even stronger form of isolation is to run the Apache server under its own operating system. It is now fairly easy to set up virtual computers within a Unix environment. It's then possible to take a snapshot of the original implementation of the operating system and then refresh the installation regularly. This means that any damage can be easily repaired.

2.4.5 Hardware isolation

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 enclosure, RAM and storage for about $150.

3 Documentation

Finally, we want the actual PHP code that implements the web-based previewer. This is included in the TEX file [2]. Running the file through LATEX prints the documentation along with instructions for extracting the PHP code.

This code has worked properly with all browsers tested (Firefox, Safari, Internet Explorer, Chrome, Opera). Nonetheless, it should be considered as a starting point. It is hoped that it may be improved by making it more robust and, hopefully, not be compromised by the types of attacks given in this paper.

References

[1] Stephen Checkoway, Hovav Shacham, Eric Rescorla. Are Text-Only Data Formats Safe? `http://cseweb.ucsd.edu/~hovav/dist/texhack.pdf`

[2] Michael Doob. A web-based TEX previewer — Sources. `http://tug.org/TUGboat/31-2/doob-texwebviewer.tex`

◇ Michael Doob
Department of Mathematics
The University of Manitoba
Winnipeg, Manitoba R3N 2T2
Canada
`mdoob (at) ccu dot umanitoba dot ca`