Grant Application for minted Development

Geoffrey Poore

12 July 2023

Abstract

I am applying for a grant to rewrite the minted package for syntax highlighting to add two features: (1) eliminate existing `\ShellEscape` security issues, which will allow minted to be used safely with restricted `\write18`, and (2) using the revised architecture required to address `\ShellEscape`, allow minted to be extended using Python in addition to LaTeX macros, which will make it simple to implement many new features.

minted overview and security vulnerabilities

The minted package for syntax highlighting of computer code was originally created by Konrad Rudolph in 2009. It saves the computer code in a `\mintinline` command or `minted` environment to a temporary file, invokes the `pygmentize` utility via `\ShellEscape` to create a syntax-highlighted version in LaTeX format, and then inserts the result via `\input`.

I took over maintenance and development of minted in 2013. I was developing the pythontex package at the time, and as part of this was already using the Pygments library and working on improving computer code typesetting. I have focused on enhancing the code typesetting aspect of minted. In 2016, I moved several improvements in minted typesetting into the new package fvextra, which extends fancyvrb by adding features such as line breaking and more robust verbatim commands. I have also made some improvements to the `\ShellEscape` and `pygmentize` side of minted, primarily by adding caching of highlighted code.

Like many packages using `\ShellEscape`, minted has always had security vulnerabilities. It requires running LaTeX with the `-shell-escape` command-line option, which enables unrestricted `\write18` (`\ShellEscape`). A document using minted can contain completely unrelated `\ShellEscape` commands, such as `\ShellEscape{\detokenize{rm -rf *}}`.

In its current form, minted is incompatible with restricted shell escape, which allows shell escape only for a specific list of trusted executables and limits the form of commands. The `pygmentize` commands are not of the correct form (though this could be modified), and minted currently relies on `rm` or `del` to clean up its temporary files. (See minted issue #271 for additional details.) Even if this was modified to work with restricted shell escape, security issues would remain. `pygmentize` allows custom lexers (language definitions) to be specified for languages that are not supported with built-in lexers. Lexers are implemented in Python, which means that using a custom lexer is equivalent to executing arbitrary code. So even if minted used `pygmentize` in a manner compatible with restricted shell escape, a document could simply include a file `lexer.py` and specify it as a custom lexer to execute arbitrary code.

Making minted safe for restricted `\write18`

I propose to create a new Python executable, perhaps `minted.py`, that works with the Pygments library directly, bypassing the `pygmentize` executable bundled with Pygments. A new minted executable can be designed in a manner that is compatible with restricted shell escape. A custom executable will also enable minted to be extended using Python rather than only LaTeX macros.

The minted executable will have the following properties:

- **Simple commands**: Commands of the form `pygmentize <options> <code_file>` are currently used by minted. These are difficult to work with from a LaTeX perspective, because LaTeX text must be properly escaped/expanded/detokenized to assemble `<options>`, and then `<options>` must be escaped yet again in a platform-dependent manner to make it compatible with the shell. From a security perspective, `<options>` involves mixing code execution with data, and any errors in validating `<options>` in a cross-platform, cross-shell manner may result in arbitrary code execution. I propose using something simpler like `minted <file>`.

- **Separation of code execution and data**: Instead of the minted executable receiving options from the command line, it will receive both options and the computer code to be highlighted from a file, like `minted <file>`. LaTeX will write options and the computer code to `<file>` using a key-value data serialization format. The minted executable will then read this data, process the computer code with the Pygments library, and save the highlighted code in LaTeX format for `\input`.

- **Safe handling of custom lexers**: Custom lexers can be used in two ways. A custom lexer that is written as a Python package can be installed, and then is available to Pygments automatically. This presents no security issues, because the user chooses to install the lexer. A custom lexer can also be used by specifying a path to a Python file. This is the problematic case, because a document could simply include a Python file with arbitrary code, and then specify it as a custom lexer. I plan to disable loading custom lexers from a path and encouraging custom lexers to be bundled as Python packages.

  I also anticipate supporting a configuration file, perhaps in the user's home directory, that enables loading custom lexers from a path. This could limit custom lexers to specified cryptographic hashes. Any configuration file will be in a location not writeable by LaTeX, outside the document directory, so that documents cannot include minted configuration files or otherwise set their own security levels.

The LaTeX part of the minted package will need minimal modification to work with the output of a new minted executable, since the executable will generate highlighted computer code in the same format currently produced by `pygmentize`. I anticipate a few minor changes to improve LaTeX error messages.

The LaTeX part of the minted package will need extensive modification to provide input to the new minted executable via serialized data rather than command-line options. This will require modifications to option handling and new LaTeX macros to serialize data and then write it verbatim to a temporary file.

Over the years, minted has had a series of issues related to package options that are not correctly detokenized or escaped for the shell. Some of these were only resolved in the most recent

release, version 2.7. (And then the fix broke several shell hacks that users had developed to add unsupported functionality!) minted has also had issues related to including external code from file paths containing spaces, or paths involving a leading tilde or shell variables. This entire category of bugs should be permanently eliminated by the move to serialized data.

Extending minted with Python

Currently, minted only uses Python to perform syntax highlighting, via `pygmentize`. Adding a minted executable will allow for more of minted to be implemented in Python rather than LaTeX macros, making possible many new capabilities. Several new capabilities will be implemented as part of the new Python executable, including the following:

- **Including part of an external file based on a regular expression, or starting/ending delimiters**. This has been requested for years. A Python implementation will be trivial, whereas a LaTeX implementation (outside LuaTeX plus a full regex library) would be difficult and limited by the capabilities of something like l3regex.
- **Official support for custom lexers specified via path**. Currently, custom lexers specified via path (that is, not installed as a Python package) are not supported. Users have developed a number of hacks over the years to add support, but these have often broken as I have gradually improved `\ShellEscape` quoting and other parts of the package. Due to the security considerations related to custom lexers (discussed earlier), they would not be supported by default. They would be enabled via a configuration file outside the document directory, likely with an option to restrict permitted lexers via cryptographic hash.
- **Including external files when file paths include spaces, leading tildes, or shell variables**. Proper path quoting and expansion is a longstanding issue. Moving from command-line options to serialized data should permanently eliminate this class of issues while providing more flexibility.

The following list of capabilities are not planned as part of this proposal (so that the requested grant amount is smaller), but are examples of the types of features that become possible with a minted executable.

- **Improving Pygments features like** `escapeinside`. The `escapeinside` option allows computer code between two designated characters to be treated as LaTeX rather than computer code. It is useful for inserting things like LaTeX symbols or math snippets. The current Pygments implementation is fragile and can fail to behave as expected depending on the details of a lexer's tokenization. A minted executable would provide a location for experimenting with alternative implementations of features like `escapeinside` outside the Pygments codebase.
- **Other highlighting libraries**. minted only uses Pygments to provide code highlighting. A minted executable would allow other Python libraries and perhaps other programs to be used instead. For example, Pandoc includes built-in support for exporting highlighted computer code in LaTeX format. A minted executable could run Pandoc in a subprocess to obtain the highlighted code, then perform any desired postprocessing. Because Pandoc can read from stdin and write to stdout, this should be possible without introducing any additional security considerations.

- **Separating highlighting from compiling**. Currently, minted highlights code during a single compile, using `\ShellEscape` to process each uncached snippet of computer code. An alternative would be to collect all code snippets in a data file during compiling, then process this data, and finally `\input` the highlighted code during a second compile. This could result in much better performance for a long document that is only built once or has no existing cache, since it removes the overhead of starting multiple shells and multiple `pygmentize` instances. A minted executable should make implementing such capabilities a logical extension of the new architecture.