Cowan, Crispin, Matt Barringer, Steve Beattie, Greg Kroah-Hartman, Michael Frantzen, and Jamie Lokier. "FormatGuard: Automatic Protection From printf Format String Vulnerabilities." In *USENIX Security Symposium*, vol. 91. 2001.

Format Guard:

FormatGuard is a solution to format string vulnerabilities like the ones we studied in class. It is a patch of glibc where the C language’s format string parsing functions. An example of one of these common functions would be printf. The danger that comes from format string vulnerabilities is that simple % directives in tandem with a lack of argument counting can change the memory of the program. We choose not to rewrite these C functions because of the plethora of legacy code that would be severely impacted by the change. FormatGuard is an additional function that you add to the stdio.h file and it effectively counters the issue by providing a way to compare the number of arguments presented to a printf and the number of arguments called by the format string. They had amazing results with a false positive rate of 0% and a performance penalty of 20%. We should care about this patch because it can solve a prevalent vulnerability in current and legacy software. Tools like this one can prevent catastrophic results from a simple mistake by acting as a safe guard. The only work you must do is add in the lines of code that protect the vulnerable functions.

Shankar, Umesh, Kunal Talwar, Jeffrey S. Foster, and David Wagner. "Detecting format string vulnerabilities with type qualifiers." In *USENIX Security Symposium*, pp. 201-220. 2001.

This is another approach to mitigating the problem of format string vulnerabilities. Instead of shoring up the exiting functions it scours the program for “tainted” input or user input and marks the variables and memory it touches. The most important feature is that it throws an error or alert if this data is passed to a format string function like printf. It detects these errors with a constraint-based type-interface engine. It is a static taint analysis, so it is performed before the code is even run and prevents the damage before it occurs. This tool is perfect for examining legacy applications that were written without the format string vulnerability in mind. It has a decent looking UI that affixes to the terminal window you are editing in, but it is by no means a sleek display. Results indicated that it was able to found all known bugs and some bugs that were not known to the code authors. The analysis normally took less than a minute and had an incredibly low false positive rate. This product is important to our research in demonstrating that you can write software that can scan code effectively for security vulnerabilities. Hopefully we will be able to build on this research to extend the use of type qualifiers to other known vulnerabilities.