

# Firefox Extension Development Tutorial :: Security

## Table of contents

1 Writing Secure Code.....	2
2 Prevention.....	2
3 Bugs, Flaws, and Vulnerabilities.....	2
4 Lessons from the Greasemonkey Security Flaw.....	3
5 Important Concepts.....	4
6 Further Reading.....	4

## 1. Writing Secure Code

Anytime an application (such as Firefox) allows user code to execute without being validated can open up security vulnerabilities. As a developer it is your responsibility to ensure that your extension does not open any vulnerabilities to the Firefox browser.

While most extension code is JavaScript, it has significantly more permissions than ordinary JavaScript found on a website. For example, JavaScript within an extension that is registered with the Mozilla Chrome registry has access to the local file system. When building a new extension, you should take into security concerns from day one.

Users should also be cautious when downloading and installing extensions. Fortunately, now that you know how to develop extensions, you can spend time reading through the source code of an extension before you choose to install it on your computer!

## 2. Prevention

Maintaining a secure browser is important business. The Firefox source code is maintained by a large number of developers and reviewed by many security experts. However, a particular extension is less likely to be reviewed by as many people. Unfortunately, a vulnerability in an extension is also a vulnerability in the browser that exposes all users to potential harm.

Once attacks are developed for a vulnerability they spread rapidly over the Internet and can often be used for nasty things such as identity theft. While more attention is generally given to security after something bad happens, an ounce of prevention is worth a pound of cure (Building, 19).

After you have finished your extension, you should monitor the various security reporting mailing lists and databases, including Bugtraq, CERT Advisories, and the RISKS digests. Look for vulnerability reports for similar extensions and preemptively fix similar problems in yours.

## 3. Bugs, Flaws, and Vulnerabilities

To understand the risk in any given software system, it is important to consider how vulnerabilities are identified and exploited. Problems arise in the presence of a software bug or design flaw, which are often the result of inaccurate assumptions. There is an every growing volume of online documentation and some very good books on topics of exploiting software and building secure software. It is important to be familiar with these topics and learn from historical exploits because "related programming errors give rise to similar exploit

techniques" (Exploit, 38).

A software exploit takes advantage of a *vulnerability* to cause the program to do something that it was not intended to do. For example, it may allow an attacker to gain access to a user's private information and make changes to someone else's computer system. One of the most dangerous forms of attack is called an arbitrary code execution, which can allow spy-ware and viruses to be installed and many nasty things.

Bugs and flaws in a piece of software cause vulnerabilities. A *bug* is a software problem that is caused by simple implementation problems, such as not explicitly initializing a variable or misusing a system call. Whereas a bug can often easily be fixed by code modifications, a flaw is a much deeper problem. A *flaw* is a design decision that can be exploited despite implementation.

#### 4. Lessons from the Greasemonkey Security Flaw

Greasemonkey is a popular Firefox extension that allows you to create scripts that can add new features to a page, remove features from a page, fix a page that is broken, or add data from other websites to a page. For example, a lot of people use Greasemonkey to insert a Delete button next to the archive button in Gmail. Some of the GreaseMonkey user scripts affect specific sites, some can affect other extensions, and some can apply to all sites. This kind of powerful manipulation has serious security implications and you can totally screw up the visuals of certain websites. However, the developers of the extension really didn't consider the security angle too much. They had a real relaxed attitude about it. Aaron Boodman, one of the developers, said: "User scripts run in content's security context. I don't see any possible problems". However, Mark Pilgrim, one of the main early adopters of GreaseMonkey discovered a huge security hole that would allow data to be leaked to remote sites.

This was a huge embarrassment initially to the developers of GreaseMonkey as well as the open source community. It became a big deal and users were advised to downgrade their version of Firefox. Slashdot also ran a story about the whole ordeal. The open source community had also traditionally touted that open source software was inherently secure because the open source process makes it so through collaborative culture that makes people check each other's code. Even though Firefox itself was not the culprit, the extension itself exposed way too much of users' computers via poorly-implemented APIs.

There were several lessons to be learned from this development debacle. Firstly, just because the overall browser is secure doesn't mean the extension automatically is going to be secure. A disciplined approach to reducing the attack surface area is necessary. Secondly, after a vulnerability has been exposed, being open about it and confronting the issue head on allows information to be given to those who need it, such as developers, IT folks, users, and security

pros. This was one of the nice things about the way Boodman handled the flaw in his extension. He updated the Mozilla extension website to warn users, conducted an open mailing list, and sought advice from people who were willing to help. Your reaction after finding a flaw is also as important as taking measures to prevent them.

## 5. Important Concepts

The following outline includes some important security concepts, without explanation. All of these topics are covered in depth in *Building Secure Software*. It is a good idea to look through the Internet and the resources listed at the end of this article.

- Principle 1: Secure the Weakest Link
- Principle 2: Practice Security in Depth
- Principle 3: Fail Securely
- Principle 4: Follow the Principle of Least Privilege
- Principle 5: Compartmentalize
- Principle 6: Keep it Simple
- Principle 7: Promote Privacy
- Principle 8: Remember That Hiding Secrets is Hard
- Principle 9: Be Reluctant to Trust
- Principle 10: Use Your Community Resources

## 6. Further Reading

- [Exploit] Greg Hoglund and Gary McGraw, *Exploiting Software: How to Break Code*. Addison-Wesley Professional, 2004.
- [Building] John Viega and Gary McGraw, *Building Secure Software: How to Avoid Security Problems the Right Way*. 2001: Addison-Wesley Professional.
- [Mozilla Security](#)
- [Security Focus](#)
- [SecurityFocus on Handling of Greasemonkey Security Flaw](#)
- [Firefox Greasemonkey Extension Disclosure of Sensitive Information](#)
- [CAN-2005-0399 - Firefox/mozilla gif extension heap overflow](#)