What are the most common security mistakes programmers make? [closed]

There are the obvious wtfs that make the headlines such as SQL injection, authentication in JavaScript but are there other more fundamental and common errors programmmers tend to make when writing applications without considering security?

security

edited Sep 19 '09 at 21:14

community wiki
5 revs, 3 users 67%
Quibblesome

closed as not constructive by sarnold, casperOne ♦ Feb 15 '12 at 1:21

As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this question will likely solicit debate, arguments, polling, or extended discussion. If you feel that this question can be improved and possibly reopened, visit the help center for guidance.

If this question can be reworded to fit the rules in the help center, please edit the question.

35 Answers

1 2 next

There are a lot of good tactical suggestions already. Let me offer a strategic observation. Most applications are written from the default security perspective of "allow all". This means a programmer will start coding everything wide open and then (hopefully) will start to consider elements that need to be secured (these tactical suggestions which have already been made are terrific examples of that).

This happens across the board. Everything from operating systems to fat clients to web-based thin clients are constructed this way. That's a primary reason why every Tuesday Microsoft comes out with a set of patches. Vulnerabilities continue to be discovered and must be remediated in a never-ending stream of patches.

My strategic suggestion is this: start coding from a default perspective of "deny all". Every architectural element, every layer, every object, every method, every variable...construct them to be inaccessible to anything unless you expressly allow it. This will slow down your productivity a little (at least at first). However, once you become accustomed to thinking this way, your delivered code will be vastly more secure.

Another analogy to this is when a programmer decides that unit testing is a good thing. Or maybe even TDD if you want to go to the extreme end of that spectrum. It takes a truckload more work to write the unit test first and then write the code to make the test pass than it takes to just write the code. But the end result is an order of magnitude more stable, and one could argue that overall less time is spent tracking down defects when a smart investment in unit testing is made.

answered Nov 28 '08 at 15:05

community wiki
Ed Lucas

6 Precisely! Assume all inputs and actions are hostile, and treat them as such in code. – Robert K Jun 17 '09 at 12:45

5 Is it just me? But I found this answer void of practical advice and I'm thinking "so..?" after reading it. – kizzx2 Jan 14 '11 at 16:06

The first three that leap to mind are:

1. Not validating ***every*** bit of data from the client
2. Not considering filesystem permissions when working with files
3. Leaving database access too wide-ranging (read/write when a user only need read access)

Another one that's not so much a security issue, although it is security-related, is complete and abject failure to grok the difference between hashing a password and encrypting it. Most commonly found in code where the programmer is trying to provide unsafe "Remind me of my password" functionality.

edited Nov 28 '08 at 22:00

community wiki
2 revs, 2 users 92%
ZombieSheep

2 When he says EVERY bit of data, take that literally. I have heard of people placing SQL injection attacks into third gen bar codes, like semacode, where non-numeric characters are allowed. – Bratch Dec 3 '08 at 15:58

3 +1 for not trusting user input. And please, salt those hashes. – Thanatos May 25 '09 at 2:52

Storing passwords, for that matter, any other sensitive information, as clear text.

answered Nov 28 '08 at 14:24

community wiki
mattruma

3 You need to differentiate between, for example, using hashed passwords fields in a users login field (always a good idea) and using some kind of obfuscation scheme which doesn't add security (not a good idea). "Another lesson is about security by obscurity. [...] All .fetchmailrc password encryption would have done is give a false sense of security to people who don't think very hard. The general rule here is: 17. A security system is only as secure as its secret. Beware of pseudo-secrets." - Eric S Raymond catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/… – Stefan Kangas Feb 16 '10 at 18:34

1. Thinking, that any case cannot exist. You have to consider EVERY situation.
2. Don't think that this will never happen. Murphy's right behind you.
3. Thinking the user sees the application like the programmer sees it.

edited Sep 29 '12 at 11:56

community wiki
2 revs, 2 users 86%
guerda

41 Re: #1 - "A good programmer looks both ways before crossing a one-way street." – Dave Sherohman Nov 28 '08 at 17:14

Here's what I can come up with:

- Failure to review code. Needless to say, most of the insecure code written today, exists because of the absence of code reviews by the right people. Code reviews for security are different from code reviews for functionality.
- Assuming that you wont be attacked. "Hey! We're behind a firewall, right? And this is a protected network."
- Forcing users to use the application in an insecure manner. For instance, most banking sites never displayed the browser's chrome until phishing sites sprang up.
- Not understanding trust boundaries. Often, this is seen as blind trust imposed on client-side input by the server-side code.
- Lack of clear separation of concerns - the client ends up assuming the role of the server. This is often seen as passwords being validated in JavaScript, or privileges being enforced at the client etc.
- Ignoring the scenario of request tampering. Worse, arguing that the said requests cannot be tampered with.
- Ignorance of the privileges required in the runtime environment. This often manifests as "notes" in the documentation on what privileges are to be provided, without sufficient details on why they are required. I've come across an applet that required read+write privileges on the entire filesystem to be granted to any codebase (even ones downloaded from the Internet), just to upload a file to the server.
- Insecure defaults. For instance, the system administrator password should be set manually after installation to a strong password since the installation did not cover the scenario of weak administrator passwords. Yes Microsoft, I'm looking at you.

- Home grown cryptography. Bill just noted this. Just because the smartest developer in the company cannot crack does not mean that an attacker cannot. Good security is built on the foundation of years of effort in the field by mathematicians and scientists. Only a fool would think that he can come with something better, while under a bathroom shower.

edited Nov 29 '08 at 0:53

community wiki
5 revs
Vineet Reynolds

Using directly **GET or POST parameters** in the code without validating it's content before using it.

answered Nov 28 '08 at 14:45

community wiki
Patrick Desjardins

I've seen a number of cases where where input validation is done exlusively with JavaScript on the client.

answered Nov 28 '08 at 14:30

community wiki
Jack Ryan

1 Most def. It's very common for beginning developers not to realize how easily client-side validation can be hacked. The golden rule is client-side validation is for usability only. For security, all input must be validated on the server-side, always. – urig Nov 28 '08 at 22:51

Not knowing what a threat model is. Given there is an endless amount of security to be applied it's vital to know where to start given your setup and business.

answered Nov 28 '08 at 14:45

community wiki
dove

Not thinking about security when designing the application..

answered Nov 28 '08 at 16:47

community wiki
user35978

1 I really don't agree with that down vote. Security is something you need to think before coding, not after. And from my experience, programmer usually code the application, and then, makes it safe. – user35978 Nov 29 '08 at 1:58

2 Agreed. It's 10X harder to add security to an existing application than to design it in up front. – Bill the Lizard Nov 29 '08 at 3:49

Technical Debt

answered Nov 28 '08 at 21:08

community wiki
Hugo S Ferreira

Programmers *trust* too much! It's not just data from the user that should be checked, it's anything from a database, filesystem, configuration file, anything that's external to the application.

answered Nov 28 '08 at 16:03

community wiki
Richard Everett

Using security through obscurity -- if any part of your security strategy includes the phrase "Nobody will ever think of looking *here*!", or "Nobody will ever think of doing it *this* way!", it's not secure.

edited Dec 10 '08 at 2:51

community wiki
2 revs
Graeme Perrow

Thinking that they can consider every situation.

This problem takes many forms; from C programmers thinking they can be careful enough to use stdio or the standard string library, to C++ programmers thinking all they need is safe pointers; to PHP programmers relying on the define/include hack; to Perl programmers thinking `-T` is enough.

Protecting against security mistakes requires that you plan for what happens when it fails; What is the impact if *this* module is broken?

It's real simple: If you don't run any part of your application as root, then *a user cannot exploit your application to get root*.

Many programmers make a similar err, by thinking that some applications cannot be modularized and segregated, and that *of course*, they happen to be working on one of those very applications.

answered Nov 28 '08 at 14:43 — community wiki geocar

I would add these:

- Lacking proper understanding from a security point of view, of the underlying platform upon which to build the application;
- Using a third-party library hastily without properly investigating or understanding the consequences the use of some parts of such a library may or may not have---this usually happens when we are in dire need of some functionality and find a third-party library that provides it, and, having found it, are in a hurry to use it without actually spending ample time going through the complete docs for the library;
- Being clueless about application and platform security in general, and how lack thereof affects your application, in particular;

answered Nov 28 '08 at 14:56 — community wiki ayaz

What? No one mentioned SO's friend BufferOverflow! It's second to cross site IMHO.

answered Nov 28 '08 at 14:56 — community wiki Robert Gould

1 You could always use a language which prevents overflows from happening in the first place... – Spence Jul 2 '10 at 3:17

Some specific gotchas--rather than the usual generic "watch out for CSRF" advice--for people building AJAX-powered sites:

Returning live JavaScript with user data. (Any site can include your endpoint as a SCRIPT tag and act on behalf of your user, if he or she is signed in. Examples: Facebook, 30Boxes.)

Failing to disallow GET on URLs that should only accept POST. (Any site can include your endpoint as an image and act on behalf of your user, if he or she is signed in. Examples: Netflix, Amazon.)

Returning an HTML page instead of JSON data if the user is not signed in. (Any site can include your endpoint as a script, watch onError, and tell if your user is signed in. Examples: Twitter, Google, and many, many others.)

answered Nov 28 '08 at 17:18 — community wiki Kent Brewster

Check out OWASP Top 10

answered Dec 20 '08 at 1:35 — community wiki dr. evil

The most common security mistakes I see come from people trying to protect from external threats and completely ignoring internal threats. You can have the most secure computers and databases in the world, but if the sensitive data is in a print-out on your desk, you're an easy target.

Think about the recent news stories about unauthorized phone company employees accessing Obama's call history, unauthorized US State Department employees accessing Senator Clinton's passport file, ChoicePoint employees stealing customer information, and so on. Everyone likes to trust the people on the inside.

I once had to work with a third party to install secure machines in a bank. It wasn't my choice, but that's how it is. The machines were compromised in two minutes because a random guy in the machine room asked them for the root password and they gave it to them, despite explicit instructions to give it to no one but me. That was just inexperience and incompetence, but it violated the bank's requirement for security, and it was all internal.

edited Apr 10 at 20:36

community wiki
2 revs, 2 users 89%
brian d foy

Designing security with too many "gates".

Imagine security as a fence with gates with guards at the gates. The best security is a very high fence or under a mountain, with a single path inside and one or more gates on the path with dedicated, paranoid guards.

The other way is a PHP web application where every individual PHP file has code to check the user authorization cookie. This code is trivially easy to forget or to get the permission check wrong. This is a bunch of gates side by side without a fence and some of the gates are unguarded.

answered Nov 28 '08 at 22:25

community wiki
Zan Lynx

Biggest mistake is to throw security issues out of scope of system development and put it on shoulders of everybody else. Common practice is to state that security is a responsibility of system administrators. Quite common also is blaming clients for low attention to security and vogue requirements regarding this. While there is a point of course it is IT department responsibility to elevate this topic and draw attention to the question.

answered Dec 3 '08 at 20:42

community wiki
Din

Thinking there are programmers who don't need to care about security.

In some circles: Using String concatenation with user-supplied data to build SQL statements.

answered Nov 28 '08 at 14:27

community wiki
Joachim Sauer

The fundamental problem is that ... programmers think security of a application is only in its login page asking "login name & password"

This ignorance is the big mistake the programmers make.

answered Nov 28 '08 at 16:13

community wiki
Murthy

---

1. Thinking it will never happen to you!
2. Not sanitizing user input
3. Not requiring strongish passwords (you can be secure without irritating your users)
4. Not using the compiler or run time effectively (windows xp sp2 had bits recompiled using the C++ compiler's /GS switch to help with buffer overruns)

answered Nov 28 '08 at 16:58

community wiki
Chris

Probably some of the most common errors are:

- trying to identify and block bad stuff, instead of only allowing good stuff (a.k.a. allow by default instead of deny by default). It's a lot easier to identify the 3 or 4 things your code is supposed to do, and code to make sure it only does those, than it is to identify the zillions of things it isn't supposed to do and code to stop them happening.
- treating security as something to be added later, instead of designing it in at the start. For example, thinking that security is something that can bolted on at the end with encryption or a firewall, or a penetration test, rather than something that is part and parcel of the whole design and QA process
- implementing your own authentication/encryption/random number generator rather than using a proven system
- assuming an attacker will be running your code and will only do things you expect - e.g. that the client accessing your server will be the one you wrote, rather than the one anyone can write.
- trusting too many things, leading to transitive trust issues. e.g. when you have component A trusts component B trusts component C, then any hole in component C is a hole in the entire system. Part of your thought process should be 'normally we control component C, but what if an attacker owns component C?'.

answered Dec 14 '08 at 2:19

community wiki
frankodwyer

Using the SA account to talk to a MSSQL Database while just execute rights on sp's should be enough.

Assuming that since you are on a closed system security is not important since it is not connected to the internet. Like internal users never do anything wrong or bad.

answered Nov 28 '08 at 16:15

community wiki
Coentje

Improper Error Handling I've seen this several times. The app crashes for whatever reason, and in order to have a clue of what happened the developer leave the stacktrace reachable to the end user.

It is not enough to show an "Try again later" if there's a detail button that shows the stacktrace. With this information the end user may know the technology, API, library, version, structure, even database information, that eventually may lead to steal information.

The best practice for this is to generate a log number, and provide that number in the detail. If a report is needed that number could be provided and used to research in the log files

A good example of PROPERLY handled error is SO Kitty "Wait a minute, I'll fix it!".. plus is cute.

answered Nov 28 '08 at 20:29

community wiki
OscarRyz

Implementing security themselves is one of the most common mistakes programmers make. Even security professionals get it wrong sometimes, and I think a lot of programmers don't understand how hard security is to get right. Using Open Source security software is free and cheap (in terms of time). Implementing it yourself takes time, and you're almost certain to get it wrong.

answered Nov 28 '08 at 21:50

community wiki
Bill the Lizard

Creating a backdoor for administration purposes with an unique hardcoded password.

answered Jun 17 '09 at 12:42

community wiki
jmservera

A big mistake programmers make regarding security is thinking that security can be "added", 'ok, now we have to make this secure!'. This is wrong!

Security is built from archicture and then in designing the system. Once you have arrive to coding, most of the security's fate of the project is already written!

answered Jan 15 '10 at 13:30

community wiki
Ariel

Not handling every returned error code, especially for system calls, and not handling exceptions. Forgetting to check the value of errno in C and C++ programs is also another no-on.

answered Nov 28 '08 at 14:32

community wiki
littlenag

---

1 2 next