Securing the stack

With a prime focus on securing your database

Colin Charles, Community Relations Manager, MySQL AB
colin@mysql.com | http://bytebot.net/blog/

Who am I?

- Community Relations Manager, MySQL
  - Distribution Ombudsman
  - Community Engineering
  - Summer of Code
  - Forge Dude
  - Generalised Dolphin Wrangler
- Previously:
  - Fedora Project FESCO and PowerPC hacker
  - OpenOffice.org contributor

General thoughts on security

- Physical security
  - Can I access the console?
- Software security
- Accept that you ***will*** get broken into

Security 101

- Prevent access
  - `nologin`, `scponly`, etc.
- Patch software, always
- Use sensible operating systems
- SELinux *can* be your friend
- Jails, virtual machines and separation of services
- A firewall is definitely your friend
- Do you read your log files daily?

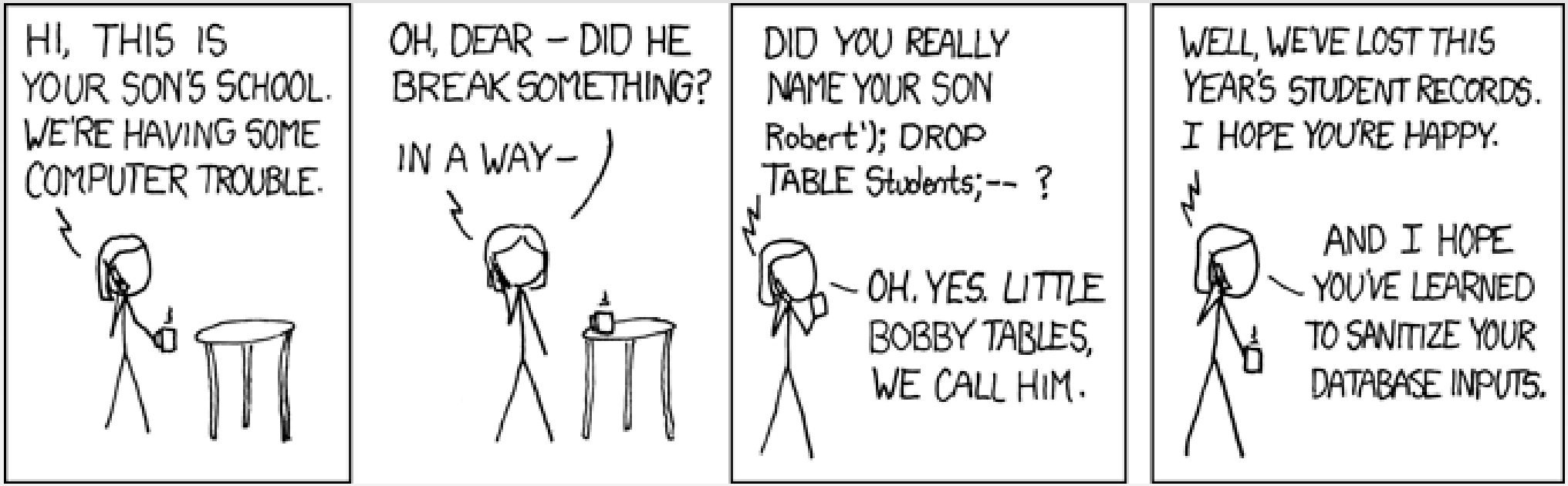
Security through obscurity?

- Disabling logins on Wordpress, MediaWiki
- SSH
  - Automated attack scripts, everywhere
    - Run SSH on a different port
  - `PermitRootLogin no`
- Ubuntu goes as far as disabling the root user – everything done via `sudo`
- Ensure only those in a certain group (`wheel`, maybe) can `sudo`

What is a database?

- Collection of records in a structured form
- A DBMS usually means:
  - Storage
  - Retrieval
  - Processing
- Security? Integrity of data?
  - Remember: information received from one record, can damage information in another

Information received in one record, damaging another



http://xkcd.com/327/

SQL injection

- User input incorrectly filtered for string literal escape characters embedded in SQL statement
- User input not strongly typed, unexpectedly executed
- `$sql = “SELECT * FROM users WHERE id = ” +number+ “;”`
- Input: `256;DROP TABLE users;`
- Parsed as: `SELECT * FROM users WHERE id=256;DROP TABLE users;`

Never trust your user-entered data

- If using PHP/MySQL, your best bet is to use:
  - `mysql_real_escape_string()`
  - http://dev.mysql.com/doc/refman/5.0/en/mysql-real-escape-string.html
  - `ext/mysqli` supports prepared statements, improved authentication protocol (5.0 and greater)
- Java: `PreparedStatement`, Ruby: `quote()`
- Protecting string data is a common thought, but remember that numeric data values need to be checked for sanity too!

Mastering the GRANT system

- Your MySQL server does have a password, right?

```
mysql -uroot
ERROR 1045 (28000): Access denied for user
  'root'@'localhost' (using password: NO)
```

- SHOW GRANTS;

```
GRANT ALL PRIVILEGES ON *.* TO
  'root'@'localhost' IDENTIFIED BY
  PASSWORD '5ds432bf13g3k274' WITH
  GRANT OPTION
```

Securing MySQL further

- REVOKE privileges as necessary
- Limit SHOW DATABASES access
- Plain-text passwords (default in Debian/Ubuntu = bad) are silly
  - Use MD5() or SHA1()
- MySQL runs on port 3306
  - telnet hostname 3306
- Ensure that port 3306 is not accessible remotely (hello firewall!)

Remember, not trusting user input?

- SQL injection, cross-site scripting, it happens to the best of us
- Script the following:
  - Enter ', “ to all input in web forms. Errors? Fix it
  - %22 (“), %23 (#), %27 (') in dynamic URLs
  - Characters, spaces, special symbols in fields
- Application should connect to database using a special, unprivileged (restricted) account

Delving into securing MySQL

- FILE privileges shouldn't be given
  - `SELECT ... INTO OUTFILE` – for security, it doesn't overwrite an existing file
  - `LOAD DATA` – can load say, `/etc/passwd`, and be accessed via `SELECT` (bad)
- `max_user_connections` – set it sensibly (you'll need more open connections for scalability, but too many and you get a denial-of-service)

Some more general thoughts

- DNS – do not trust it. When GRANTing access, and so on, use
  - 'user'@'ip.address'
- SSL support is built-in, use it
- Transmitting plain-text? Check
  - `tcpdump -i <interface> -l -w src or dst port  3306`

So?

- You've secured MySQL
- You've secured Linux
  - Firewall, SELinux, apply patches, sensible OS
- What do we save for another talk?
  - Apache?
- What about PHP? (focus on the application)
- What about other languages?

Recent war stories

- Wordpress MU (WPMU) was allowing attackers to change blog content, via the XMLRPC interface
  - Immediately do a database dump
  - Remove XMLRPC support
  - ... find, and fix the bug
- This happened on http://blogs.mysql.com/

Recent war stories II

- MySQL Forge
- Cookies, stolen via JavaScript, could potentially allow an attacker to view MySQL Intranet content!
- Solution? Searching code must not be susceptible to cracking via JavaScript
- Cookie separation is important

So, I've been broken into. Now what?

- Take a snapshot (hello dd)
- You do, backup regularly, right?
- Reinstall, cleanly, restore
- Why reinstall? Rootkits stay sometime
- Run forensics through the snapshot
  - Good chance log files disappeared, but they might still be on disk... (image)

In conclusion

- Security is about being proactive
- Security, is largely, also very reactive
- XSS has been around since 2000, yet modern code 8 years later still has this problem
- Kernel updates, SELinux, etc. help in most cases, but under 2 weeks ago, we had an interesting root-kit that worked, against all odds

Getting involved

- Is security your cup of tea?
- HeX-LiveCD
  - http://groups.google.com/group/HeX-liveCD
- OLPC Bitfrost
  - http://wiki.laptop.org/go/Bitfrost
- Write, contributed, automated tools to scan websites for vulnerabilities

Resources

- http://httpd.apache.org/info/css-security/
- http://dev.mysql.com/doc/refman/5.0/en/faqs-security.html
- http://dev.mysql.com/doc/refman/5.0/en/privilege-system.html
- http://dev.mysql.com/doc/refman/5.0/en/mysql-real-escape-string.html
- Plenty of books
  - Get an O'Reilly Safari subscription
  - Visit your favourite bookstore (MPH MidValley, Kinokuniya KLCC, Borders all have reading lounges)

Thanks! Questions?

E-mail me:
colin@mysql.com
Catch me on IRC, at irc.freenode.net, #mysql-dev / #myoss:
ccharles