Cryptonite

Top encryption pitfalls that could turn your app into crap

1. Using crypto unvetted by the global cryptographic community
    a. out of 15, 2 broke before and 1 during the first AES conference
    b. Hashing competition
        i. SHA3 competition
            1. took from 2007 to 2012 with 64 entries
            2. Keccak was the winner
            3. many of them had flaws, like collisions in round 1
            4. Bruce Schneier was a sad panda, his was runner-up
            5. paper was written in 1998, so things might have changed since then.
2. encrypting something already known
    a. rainbow tables
    b. crib (slang term for cheating) - known plaintext attack (happy birthday, hitler!)
    c. helped break enigma
3. using encryption instead of hashing where encryption is not needed
    a. passwords
    b. pins
4. storing password values poorly
    a. in the file itself (example: old MSOFFICE versions)
        i. binary editor can reveal the password
    b. ...in memory
        i. core dumps can reveal passwords (wu-ftpd example)
        ii. in lower-level langs, you can overwrite with zeros when finished
    c. ...in config files
    d. ...on same server as important data
        i. auth should be separate (separation of duties)
    e. … the passwords themselves, instead of the verifiers
        i. store proper hashes, with salts, not passwords
    f. in your code repos
        i. stored passwords left in code or settings file
5. using outdated encryption
    a. need resources to look at list (FIPS 140-3 or NIST?)
6. using weak random numbers
    a. guess digit 10001 when given the first 10000
    b. initialization vectors

7. Hardcoding encryption
    a. not future proof
    b. shouldn't be done at all, allow for upgrades
    c. See this in SCADA systems
        i. SCADA system built now are still using crap code ideals, no option for upgrades
        ii. not upgrading is no longer an option
8. Hardcoded passwords
    a. again, see 6.c above as reasons to no longer do this.
9. Replaying TCP traffic
    a. requires network access, but MITM can allow for reply of traffic to gain access.
    b. proper use of session tokens, regenerating them
    c. timestamping as well
        i. Timestamping is another way of preventing a replay attack. Synchronization should be achieved using a secure protocol. For example Bob periodically broadcasts the time on his clock together with a MAC. When Alice wants to send Bob a message, she includes her best estimate of the time on his clock in her message, which is also authenticated. Bob only accepts messages for which the timestamp is within a reasonable tolerance. The advantage of this scheme is that Bob does not need to generate (pseudo-) random numbers, with the trade-off being that replay attacks, if they are performed quickly enough i.e. within that 'reasonable' limit, could succeed.
10. remote login over unencrypted channel
    a. vulnerability scanners should be able to detect
    b. tunnel connections over TLS 1.2
11. managing passwords poorly
    a. use strong key derivation functions for hash storage - example: lastpass uses PBKDF2 with sha256 and defaults to 5000 iterations and allows user to set number of iterations.
        i. PBKDF2
        ii. bcrypt
        iii. Scrypt
        iv. argon2 (source on GitHub)
    b. one password/key to rule them all
        i. different keys and passwords for different data (CC#, auth, PII)
            1. different keys will make data harder to dump all at once

http://stackoverflow.com/questions/16891729/best-practices-salting-peppering-passwords

https://password-hashing.net/
http://www.infoworld.com/article/2923777/encryption/5-ways-developers-get-encryption-wrong.html

http://scpd.stanford.edu/search/publicCourseSearchDetails.do?method=load&courseId=1285222

https://www.cs.ucsb.edu/~chris/research/doc/ccs13_cryptolint.pdf

http://csrc.nist.gov/publications/fips/fips140-2/fips1402annexa.pdf

email from Brian Hearn (bds listener) *if time allows*