Cryptography for Software and Web Developers

Part 4: randomness, hashing, tokens

Hanno Böck

2014-05-28

- In security (not just crypto) we often need random numbers
- Examples: CSRF-tokens, one-time-action tokens, password salts, key generation, ...
- There's even crypto out there that needs good random numbers to run or it'll completely break: DSA, ECDSA (but better avoid that).
- Good randomness is hard

- OS or programming language default random functions often not secure random numbers.
- Rounding problems, conversion problems can reduce space of possible random numbers vastly.
- 2008 Debian OpenSSL bug.
- PRNGs need a seed or they don't work.
- NSA managed to make a backdoored RNG an official standard and payed RSA Inc. 10 Million $ to make it the default in BSAFE
- No way to test random numbers reliably.

- An RSA public key consists of an exponent e and a modulus N which is the product of two primes
- If you know the primes you can get the private key
- What happens if we have two RSA keys with a shared prime, e. g. $N_1 = p * q_1$, $N_2 = p * q_2$? You can break this key with the greatest common divisor algorithm.
- Some people tried this with lots of SSH and TLS keys and found over 50 embedded devices that created such factorable keys. [url]
- Linux seed sources: HD timings, keyboard strokes, mouse movements. Embedded devices often have no HD, no keyboard, no mouse.

- PHP good: openssl_random_pseudo_bytes(), PHP bad: mt_rand(), rand(), uniqid()
- JavaScript good: window.crypto.getRandomValues(), bad: Math.random() (only latest browser support window.crypto.getRandomValues())
- /dev/urandom is good if it is properly seeded. For embedded devices: Better create the keys on a desktop PC.

- So many people have wrong ideas about hashes...
- Completely typical situation: I write about cryptographic hashes, people in the comments discuss about password hashes and salting
- Hashes used in many contexts: error detection (CRC32), signatures (SHA256, SHA516), passwords (bcrypt, scrypt)
- If you use a hash function you need to know what it should do

- CRC32: Very fast, no security at all
- Reliably detects errors, but trivial to construct another input for an existing hash
- Usable only for errors and if no attacker is involved (e. g. error detection on hard disks or file comparison over otherwise secure network connections).

- Cryptographic hashes need to be collision resistant and preimage resistant
- Collision: It should be practically impossible to create two different inputs with same hash
- Preimage: It should be practically impossible to create an input for a given hash value.
- Used in many places, e. g. signatures
- Some crypto protocols need hashes and don't have collision resistance requirement (e. g. HMAC), but that's usually not something that should bother you

- In 2004/2005 big breakthroughs on hash attacks, mostly the work of a Chinese team led by Wang Xiaoyun.
- Most important results: practical collision attacks on MD5, almost practical attacks on SHA1
- 2008: MD5 attack on RapidSSL leads to fake CA, 2012: Flame worm uses MD5 attack to create rogue code signing cert
- SHA-2 functions (SHA256, SHA512) considered safe today, SHA-3 will come soon.

- Idea: We don't save passwords, we just save hashes so if our database gets stolen the attacker has no direct access to the passwords
- Attackers can brute force
- Salting makes it harder
- Security requirements for password hashes completely different from cryptographic hash functions
- Collision resistance doesn't matter, they should ideally not be fast

- glibc uses several iterations of cryptographic hashes (default SHA512) and a salt.
- bcrypt and scrypt are functions designed to be password hashes. bcrypt is designed to be slow, scrypt is designed to be slow and use lots of memory.
- There's a Password Hashing Competition (PHC), results expected in 2015.

- The importance of secure password hashing is IMHO vastly overstated.
- glibc-type SHA512, bcrypt, scrypt are all "good enough", just make sure you have a salt.
- Password hashing only gives you a tiny little bit of extra protection if your database gets stolen. But if that happens you're screwed anyway.
- Make sure nobody steals your database. That's much more important.

- Factorable RSA keys `https://factorable.net/` `http://media.ccc.de/browse/congress/2012/29c3-5275-en-facthacks_h264.html`
- Password Hashing Competition `https://password-hashing.net/`