Security Guidelines
C/C++ Coding
C/C++ Coding

What did I need to do when...

by
Thomas Biege <thomas@suse.de>

... when I want to copy Strings?

- do NOT use strcpy(3) or strcat(3)
- instead use strncpy(3) or strncat(3)
- but use them with the correct 3rd parameter (which is the size of the destination buffer)
- if available use strlcpy(3), strlcat(3), they are less error prone

... when I want to copy binary Memory?

- use memcpy(3) (POSIX standard)
- take care, the third parameter is the size of the destination buffer

... when I want to allocate Memory?

- do NoT calculate the memory size based on integers provided by the user (untrusted source)
- use an upper boudary
- only allow a size greater than zero

... when I want to do arithmetic Operations?

- integers have a limited size and can therefore overflow!
- be type-safe (even on different archs)!
- check for upper and lower boundary, always!
- type-casts from signed to unsigned can cause high values and other problems
- ask the SUSE Security-Team for help!

... when I want to do arithmetic Operations?

- Example: ‘amount’ is untrusted input

```
size_t amount; // size_t is unsigned and
               // portable between different
               // architectures

if( amount * sizeof(struct hdr) / sizeof(struct hdr) != amount )
{
        log("integer overflow occurred with 'amount'!");
        exit(-1);
}

hdr_ptr = (struct hdr *) malloc(amount * sizeof(struct hdr));
if(hdr_ptr == NULL)
{
        log("unable to allocate memory for hdr!");
        exit(-2);
}
```

... when I want to create a temporary File?

- just use mkstemp(3)
- or mktemp(1) in shell scripts

... when I want to use changing Objects?

- where? filesystem, database, threads and memory, and a lot more!
- filesystem: use mkstemp(3), open(2) with O_CREAT|O_EXCL, avoid symbolic names use descriptores whereever possible
- lock resources like files, memory, etc.
- use atomic operations (syscalls), mutex locks, etc.

... when I want to use Functions with varibale Parameter Size?

- do NOT use the untrusted input as format string tag!
- wrong: snprintf(buf, sizeof(buf), user_input);
- right: snprintf(buf, sizeof(buf), “%s”, user_input

... when I want to spawn another Process?

- clean the Environment! (clearenv(3))
- and set safe values (setenv(3))
- do NOT use system(3) or popen(3), instead use execle(3) or alike
- close all open resources (files, directories, sockets, IPC stuff, ...), use fcntl(2) with FD_CLOEXEC

... when I want to pass secret Information to another Process?

- do NOt use environment varibales. everyone (not on Linux) on the system can see them!
- do NOT use command line parameters. everyone on the system can see them!
- use IPC (pipe, unix domain socket)

... when I want to write a setuid Application?

- DROP privileges as early as possible!
- order: initgroups(3), setgid(2), setuid(2)
- do NOt trust any input and check return values!
- clean up environment as early as possible!
- use Linux’ capabilities, drop more privileges!
- ask the SUSE Security-Team for a source code review

... when I want to write a Network Service?

- try to find an existing solution that can do the same job
- do not trust any input from your socket. be paranoid!
- separate your code into logical differnent processes with different privileges
- ask the SUSE Security-Team for a source code review

... when I want to handle untrusted Input?

- check range, type, size, etc.
- use a white-list instead of a black-list for filtering
- most modern libraries (like mysql) provide a function to escape dangerous characters
- in case of a stateful protocol use a state-machine, and terminate on violations

... when I need to handle sensible Data?

- after you are done, clean up the memory using memset(3) (compiler optimization removes asm code, use memset(3) twice with different value to be set)
- encrypt data on the HDD
- avoid swapping using mlock(3)
- avoid core dumps using setrlimit(2)

... when I have to use Passwords

- only store them encrypted and make this file only readable for your application
- use a “password salt” to stop dictionary attacks
- never send them in cleartext over the wire
- force at least the usage of 6 characters, mix alpha with alphanumerical characters

... when I want to authenticate Entities?

- check for existing solutions
- be paranoid about the input provided
- do not use short-cuts in your code to avoid timing-attacks, add random delays (msec)
- delete credentials as early as possible
- ask SUSE Security-Team for a source code review

... when I want to send Data over the Wire?

- use SSL/TLS to protect your data
- never send passwords in cleartext, use at least digest authentication and a nonce (number used once) to avoid replay attacks
- if encryption is not an option, try to use at least integrity checks on the data

... when I want to send Data over the Wire?

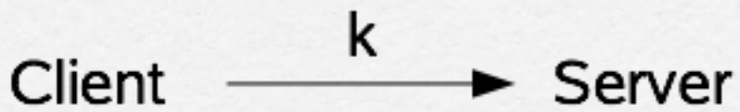
- a simple protocol for doing authentication only

The client generates a random nonce (n_client) and sends it to the server.

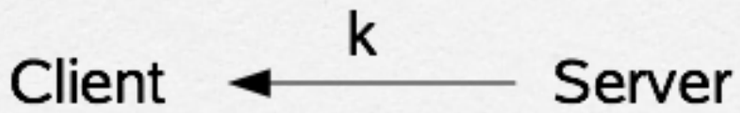
The server generates a random nonce (n_server) and sends it to the client.

Both sides compute 'n = n_client | n_server' to create a unique session nonce. (| is a concatenation, not bitwise AND)
Additionally they also compute 'Hash(Hash(Password) | n)' as session key 'k'.

.... when I want to send Data over the Wire?



The server compares the received session key with his own sessin key



The client compares the received session key with his own sessin key

This step ensures that **each** side knows the Password and that the session is fresh (not replayed). The session nonce can not be fully controlled by the attacker because it has two components. Binding this session nonce to the hashed password ensures a unique session key.
This technique avoids a reaply attack and a man-in-the-middle attack without using encryption but just a hash algorithm, a shared secret, and a random nonce.

... when I want to use SSL/TLS?

- verify the certificates!
- do NOT use exportable and/or weak ciphers!
- key length for symmetric algo.: >= 128 bit
- key length for asymmetric algo.: >= 1024 bit
- if you can not verify a certificate, show the user the whole cert + fingerprint and let him/her decide.

... when I need to provide a random number?

- use /dev/random for high security (like key generation) (... and use read(2), no buffering)
- use /dev/urandom for lower security
- never use predictable/low-entropy values like the system time!
- avoid using a PRNG of software libraries
- do NOT ship random seed with a package

.... when I want to use Cryptography?

- do NOT invent your own!!!
- use existing solutions like OpenSSL, perl-Crypt*, php(4,5)-mcrypt, etc.
- educate yourself about the solution provided, too much can be done wrong :(
- if in doubt, ask the SUSE Security-Team for help :)

... when I have Questions?

- just ask the SUSE Security-Team <security-team@suse.de> :)