Secure Programming

-

In the UNIX Environment

Thomas Biege <thomas@suse.de>

What will I talk about?

- Application Design
- Programming Errors
  - Let’s make a short Break here. (10 min)
- Cryptography and Randomness
- Secure Socket Layer (SSL aka TLS)

Application Design

A SetUID Application

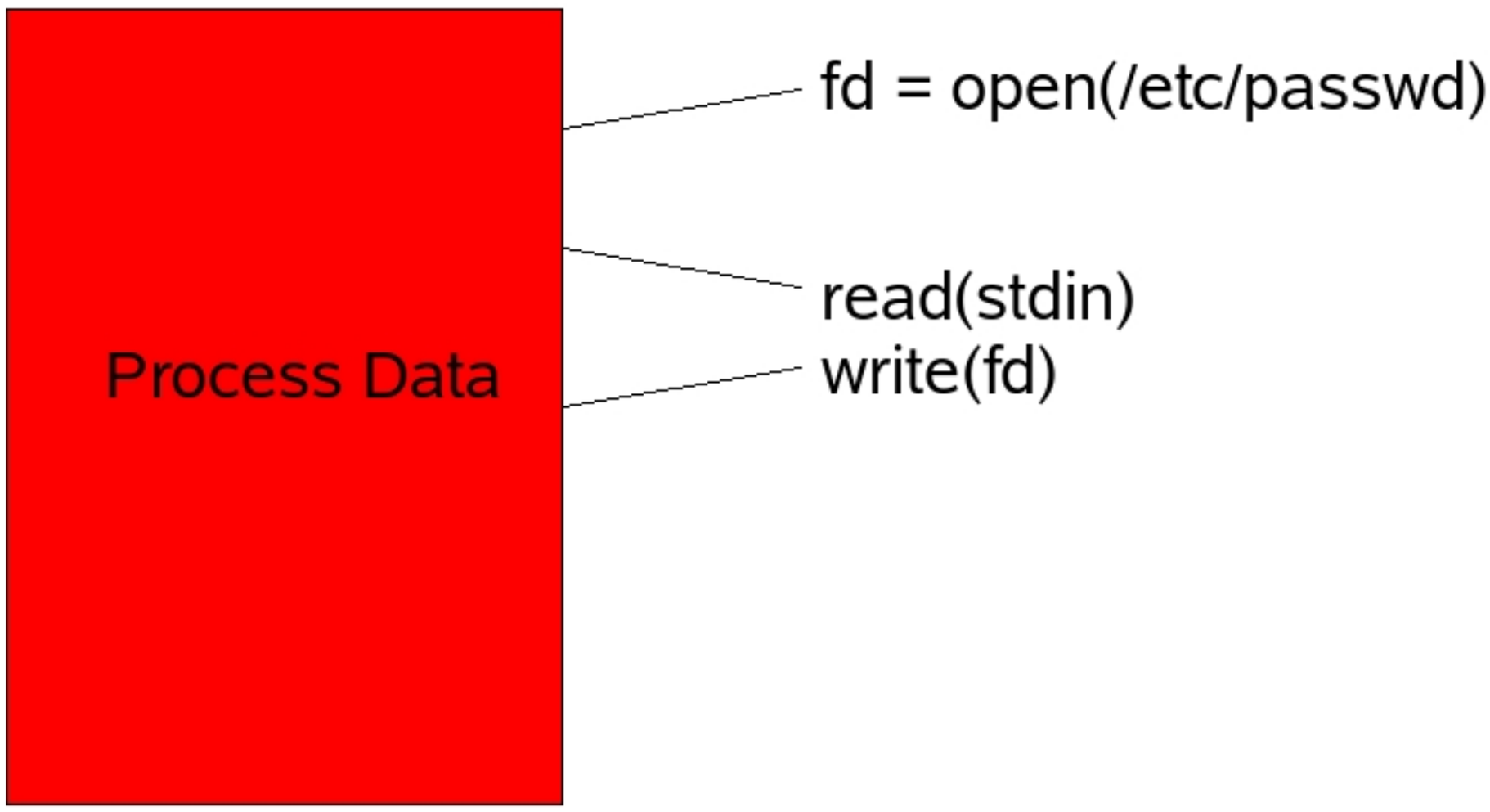
A Content Management System

A SetUID Application

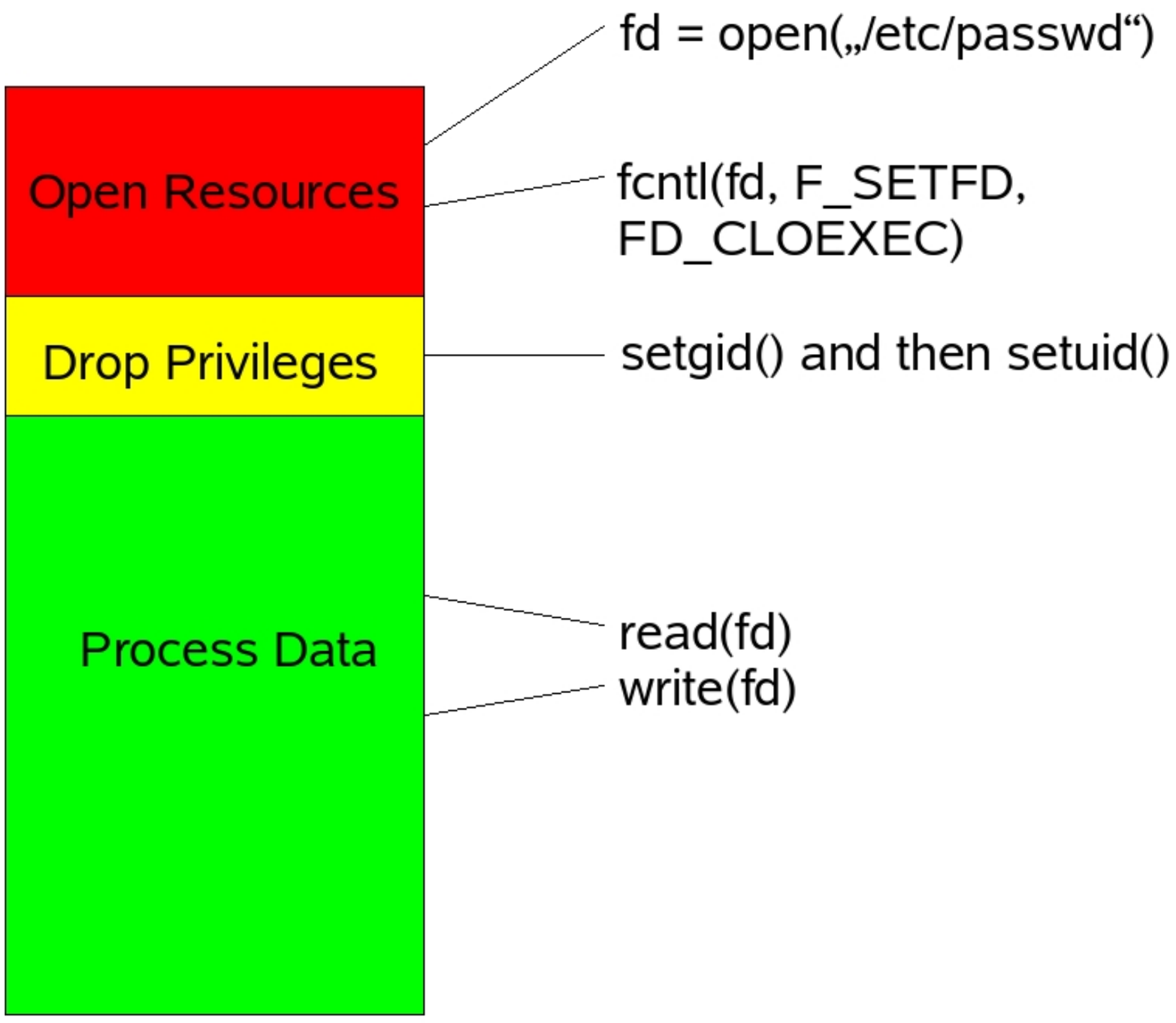
Rules

- Just don’t write it. Your Design is broken...
- and your Code will be broken too. ;-)
- drop Privileges as early as possible
- KISS
- Code Review... No, not by you.

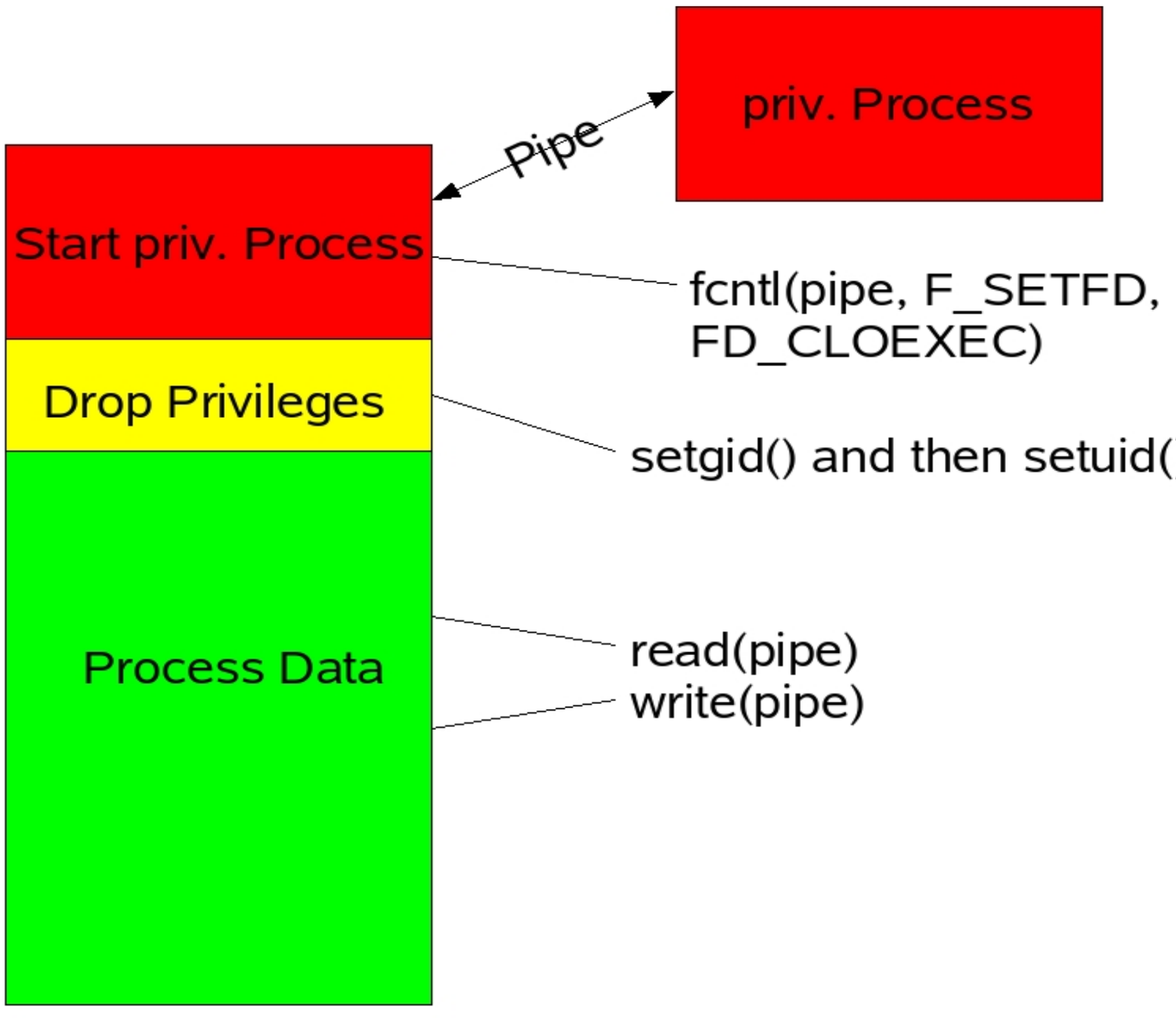
The Bad Idea



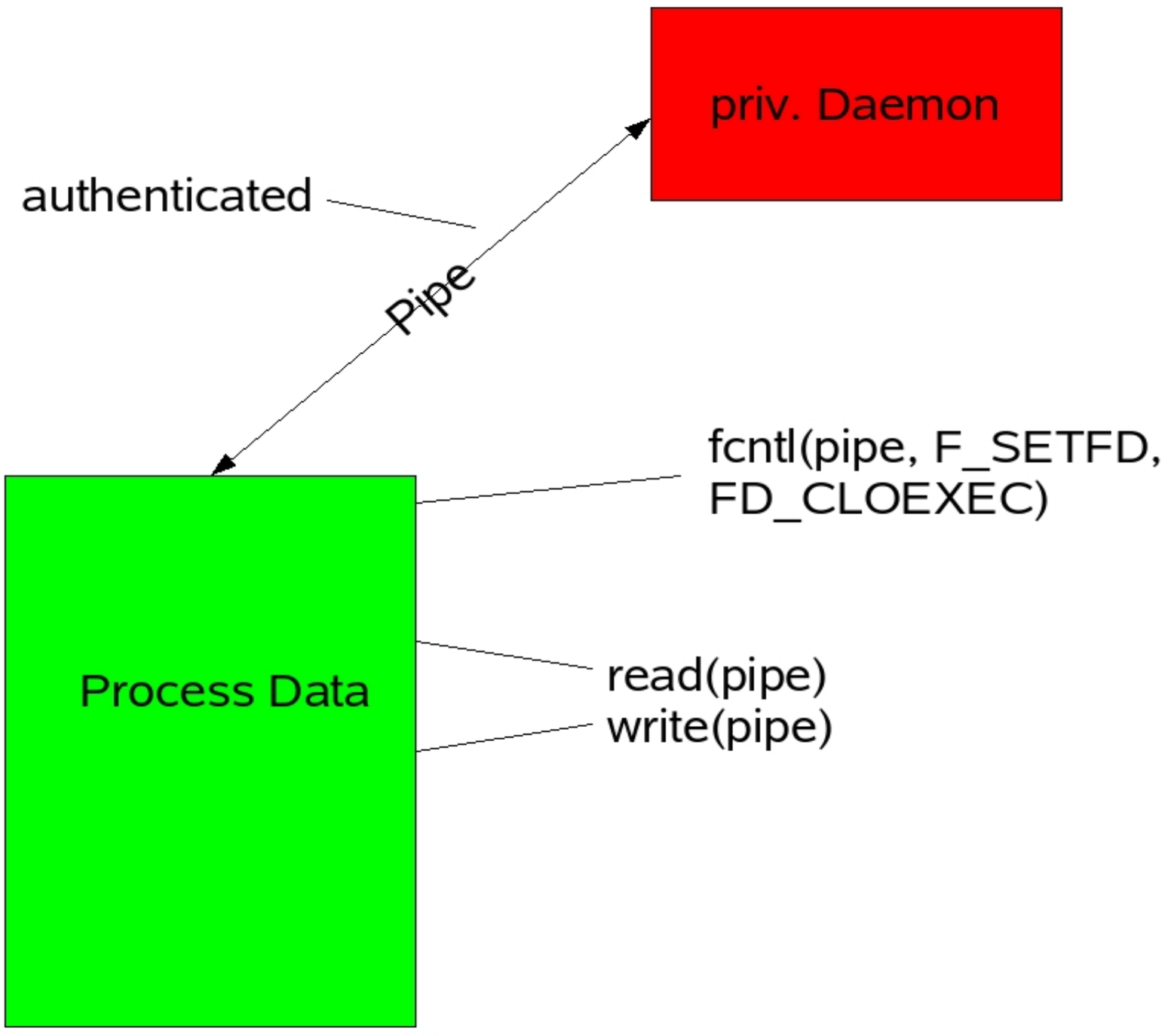
Drop Privileges



Fork a Process



Daemon

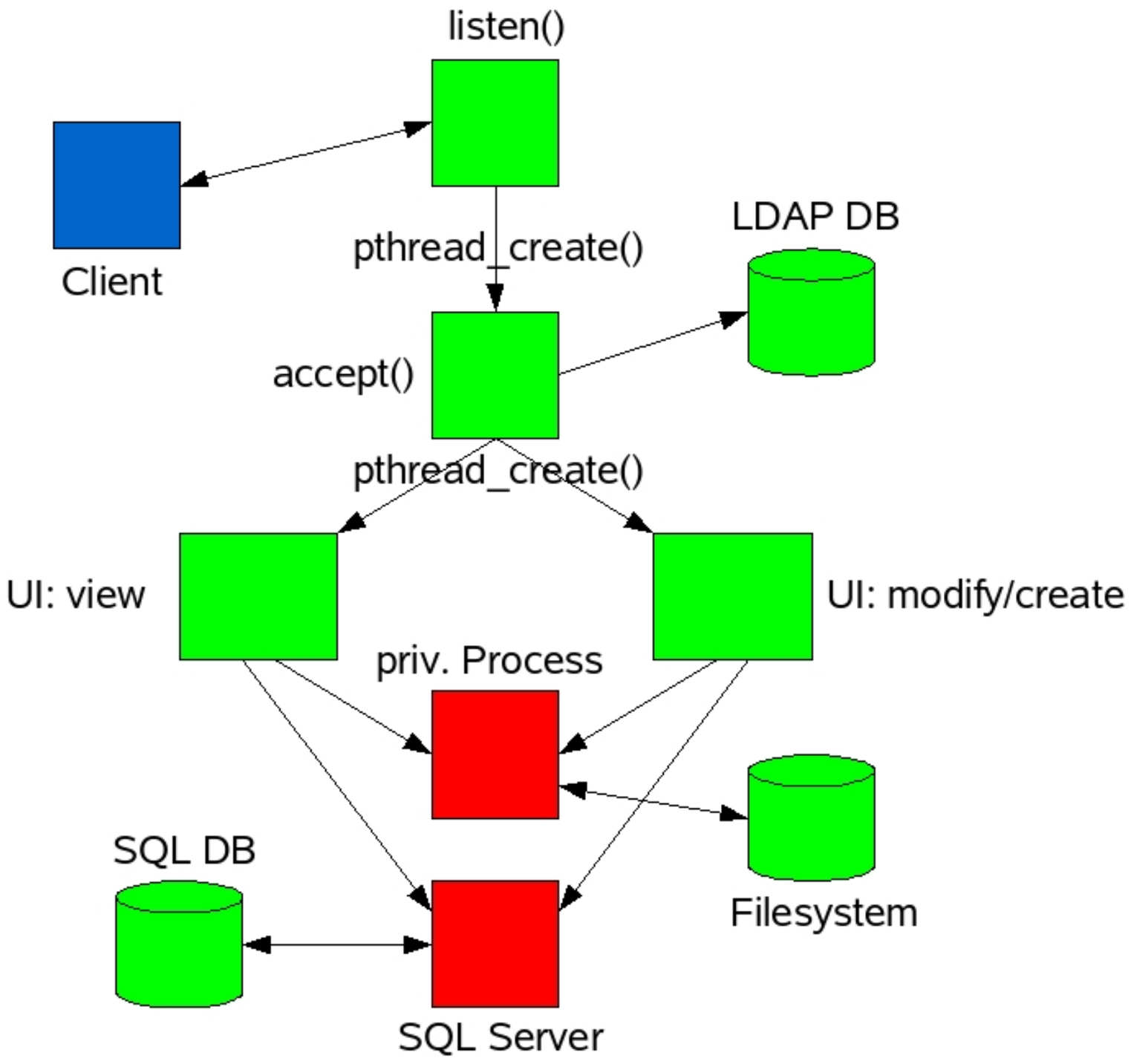

priv. Daemon
authenticated
Pipe
fcntl(pipe, F_SETFD,
FD_CLOEXEC)
Process Data
read(pipe)
write(pipe)

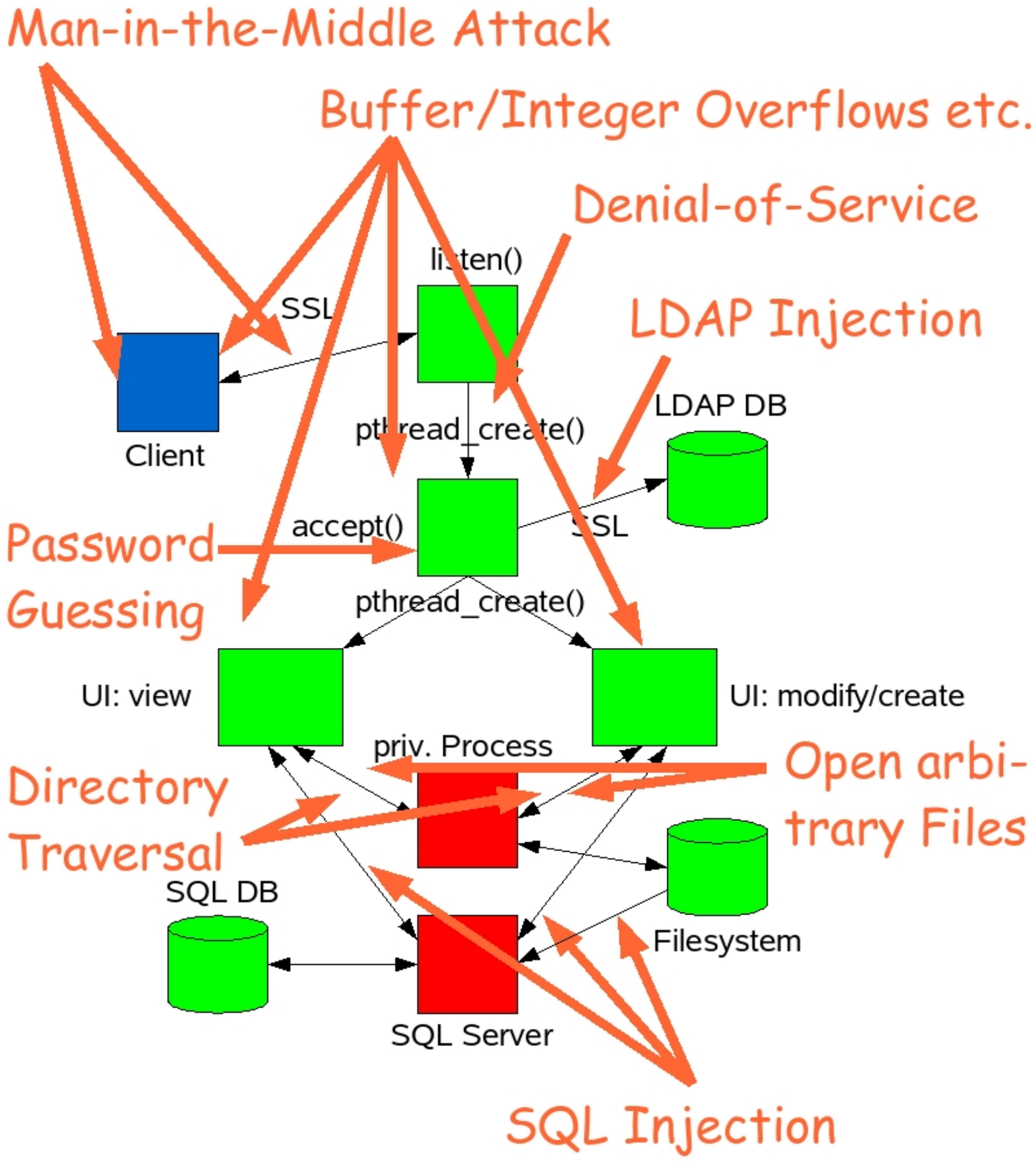
A Content Management System

Requirements

- remote Network Connections (TCP)
- a secure Channel (SSL)
- parallel Sessions
- Authentication (LDAP)
- create, open, edit HTML Files
- upload binary Files into a SQL Database
- view Documents (merge Files)

Overview


listen()
Client
pthread_create()
LDAP DB
accept()
pthread_create()
UI: view
UI: modify/create
priv. Process
SQL DB
Filesystem
SQL Server


Man-in-the-Middle Attack
Buffer/Integer Overflows etc.
Denial-of-Service
LDAP Injection
Password Guessing
Directory Traversal
Open arbi-trary Files
SQL Injection
listen()
SSL
Client
pthread_create()
LDAP DB
accept()
SSL
pthread_create()
UI: view
UI: modify/create
priv. Process
SQL DB
Filesystem
SQL Server

92

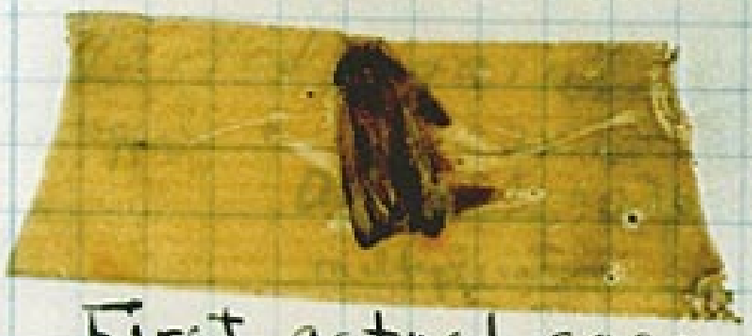
9/9

0800 Arctan started
1000 " stopped - arctan ✓ { 1.2700 9.037 847 025
9.037 846 795 correct
13 wc (032) MP - MC ~~1.9821[illegible]000~~ ~~2.13047[illegible]15 (-3)~~ 4.61592505 9(-2)
(033) PRO 2 2.130476415
correct 2.130676415

Programming Errors

Relays changed

1100 Started Cosine Tape (Sine check)
1525 Started Multy Adder Test.

1545 

Relay #70 Panel F
(moth) in relay.

First actual case of bug being found.

~~1700~~ 1630 Arctangent started.
1700 closed down.

Buffer **O**verflows

modular **A**rithmetic

Race **C**onditions

Format **S**tring **B**ugs

Untrusted **I**nput

Buffer Overflows

ziptool - *strncat(3)*

```
switch(oper)
{
    case MOUNT_DISK:
        strncpy(cmd, MOUNT_CMD, strlen(MOUNT_CMD)+1);
        strncat(cmd, " -t ext2 ", 9);
        strncat(cmd, dev, strlen(dev)+1);
        strncat(cmd, " ", 1);
        ...
```

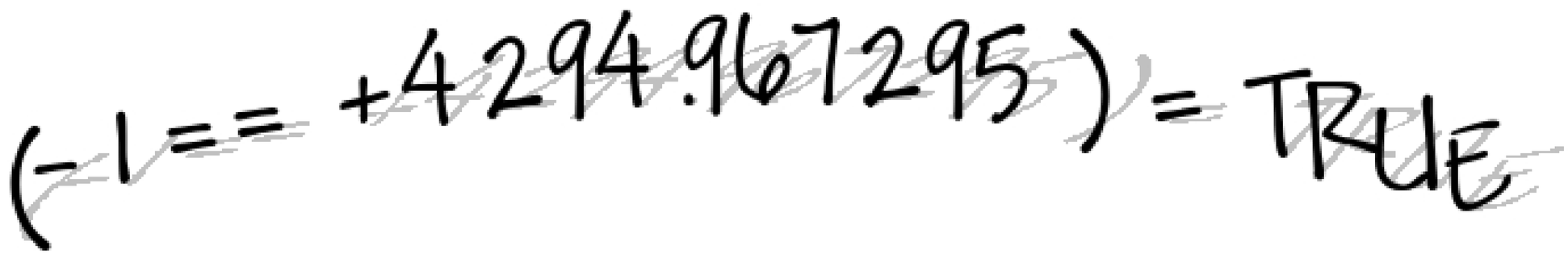
What to avoid?

- blindly using `strcpy(), strcat(), sprintf()`
- using `strncpy(), strncat(), snprintf()` with wrong Parameters
- Loops to copy Arrays w/o Bounds-Checking
- writing Functions with Array Parameter but w/o Array-Size Parameter
- trusting Input from uncontrolled/untrusted Sources... be paranoid!

Integer Overflows

Type Mismatch

Signed Issues



unsigned

1111 1111

+ 1 →

unsigned

0000 0000

---

signed

1000 0001

type cast →

unsigned

1000 0001

-32.767

32.767

---

signed

1000 0001

-32.767

type cast →

unsigned

1111 1111 1000 0001

4.294.934.529

smalltalk: Type Mismatch

```
memset (crgb, 0, 256 * sizeof (unsigned int *));
for (a = 0; a < ncolors; a++)
{
    char1 = colorTable[a].string[0];
    char1 = (unsigned char)colorTable[a].string[0];
    if (crgb[char1] == NULL)
    {
        crgb[char1] = (unsigned int *) ...
```

X11: Signed Issue?

```
unsigned int i;
int i;

for (i = nbytes; --i >=0;)
      *dst++ = *src++;
```

xpdf: 32 Bit vs. 64 Bit

```
size_t size;
[...]
if(size*sizeof(XRefEntry)/sizeof(XRefEntry) != size)
{
  error(-1, "Invalid 'size' inside xref table.");
  ok = gFalse;
  errCode = errDamaged;  #define SIZEOF_MYDATA 100
}

entries = (XRefEntry *)gmalloc(size * sizeof(XRefEntry));

for (i = 0; i < size; ++i)
{
     entries[i].offset = 0xffffffff;
     entries[i].used = gFalse;
}
[...]
```

Samba: Valgrind is evil ;-)

```
void *malloc_array(size_t el_size, unsigned int count)
{
if (count >= MAX_ALLOC_SIZE/el_size) {
      return NULL;
}

#if defined(PARANOID_MALLOC_CHECKER)
      return malloc_(el_size*count);
#else
      return malloc(el_size*count);
#endif
```

Samba: Valgrind is evil ;-)

```
void *malloc_(size_t size)
{
	#undef malloc

	/* If we don't add an amount here the glibc memset
	seems to write one byte over. */

	return malloc(size+16);

	#define malloc(s) __ERROR_DONT_USE_MALLOC_DIRECTLY
}
```

What to take care off?

- test for upper/lower Limits and Overflows
- take care: Overflows are undefined for signed Integers
- be type-safe!

Test

```
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);
}
```

Race Conditions

Where?

- Filesystem
- multithreaded Code
- distributed Databases
- Signals and Interval Timer
- etc.

RC Server (multi-threaded)

```
const char *
rcd_prefs_get_mid(void)
{
    static char *mid = NULL;
    RCBuffer *buf;

    g_free (mid);
    mid = NULL;

    buf = rc_buffer_map_file (SYSCONFDIR "/mcookie");
    if (!buf)
        return NULL;

    mid = g_strndup (buf->data, 36);
    mid[36] = '\0';

    rc_buffer_unmap_file (buf);

    return mid;
}
```

RC Server (multi-threaded)

- *Double Free* Bug due to missing Locking in multi-threaded Situation
  - Result: Crash!
- Lessons learned:
  - use Locking
  - do not use static Buffers

CASA’s Unix Domain Socket

CASA's Unix Domain Socket
The Attack

- **Attacker:** `ln -s /etc/passwd /tmp/.novellCASA`
- **CASA:** calls `UnixEndPoint(/tmp/.novellCASA)`
- **CASA:** `UnixFileInfo()` is a Wrapper for `stat(2)` and follows symbolic Links: `stat(`*`/etc/passwd`*`)`
- **CASA:** `/etc/passwd` is root-owned and therefore the Check for the Ownership succeeds

CASA’s Unix Domain Socket The Attack

- **Attacker:** removes the Link, creates a real Socket, and attaches his Daemon
- **CASA:** connects to this Socket (by Name)
- **Attacker:** steals every Secret from the Client

How to avoid them

- use Filedescriptors not Filenames
- `O_NOFOLLOW, O_CREAT|O_EXCL`
- `setgid(2), initgroups(3), setuid(2)` (Order!)
- `fork(2)` + drop Privileges
- `mkstemp(3)`
- `mktemp(1)`
- lock shared Resources while in use

Format

Bugs

- published in Year 2000
- *printf*-like Functions
- read: **`%x, %p`**
- write: **`%n`** (does not work with current *glibc*)

Wrong

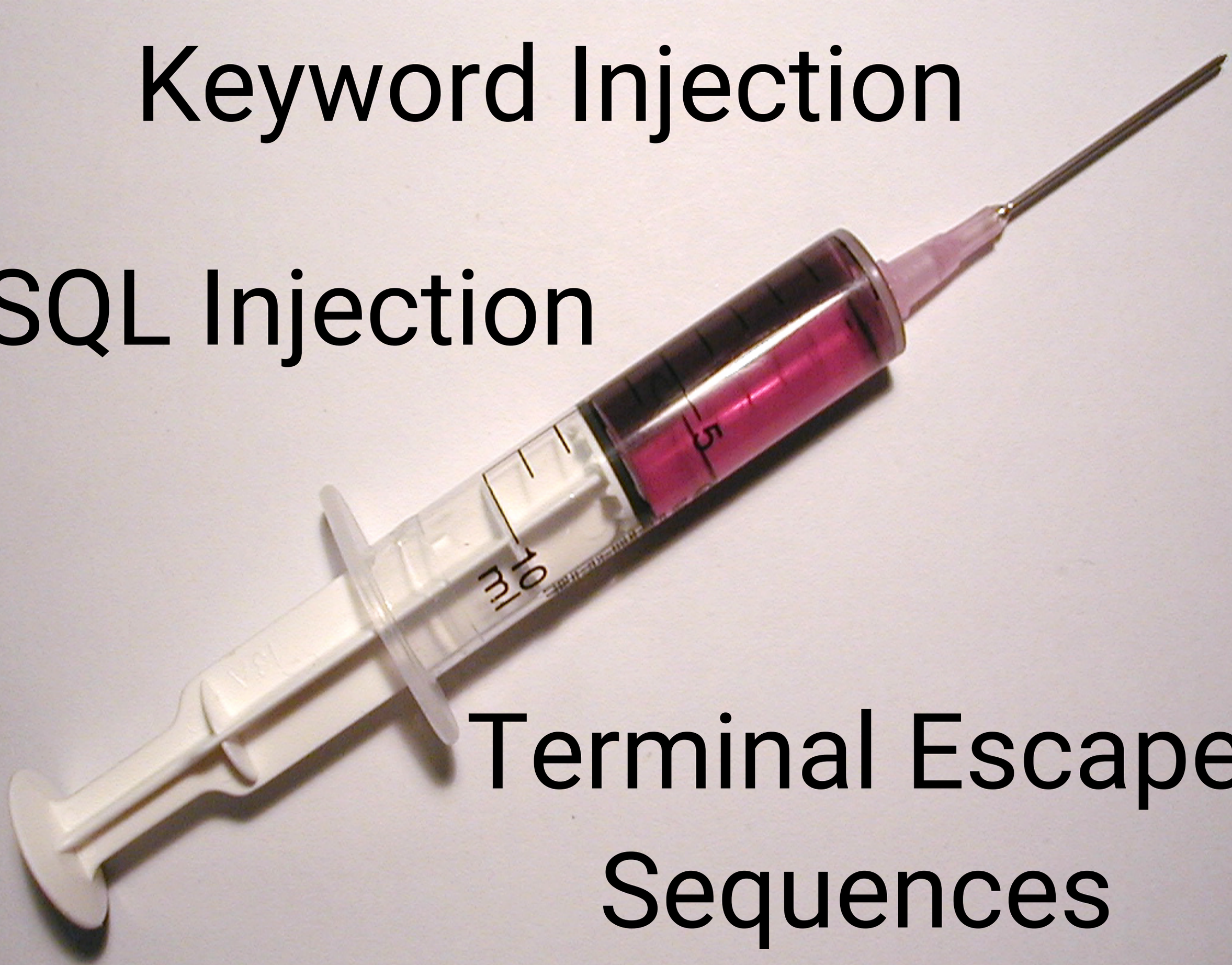
```
snprintf(buf, sizeof(buf), user_data)
```

Right

```
snprintf(buf, sizeof(buf), “%s”,
        user_data)
```

Keyword Injection

SQL Injection



Terminal Escape Sequences

ntop - *popen(3)*

```
if(num == 0) putenv("QUERY_STRING=");

sprintf(line, "%s/cgi/%s",
getenv("PWD"), cgiName);

if((fd = popen(line, "r") == NULL)
{ ...

        cgiName = ";/bin/rm -rf /"
```

AMaViS: Execute Shell

```
cat <<EOF| ${mail} -s "VIRUS IN YOUR MAIL TO $7" $2

                V I R U S  A L E R T

  Our viruschecker found a VIRUS in your eMail to "$7".
              We stopped delivery of this eMail!

    Now it is on you to check your system for viruses

  For further information about this viruschecker see:
                  http://aachalon.de/AMaViS/
          AMaViS - A Mail Virus Scanner, licenced GPL
EOF
```

Ok, what should I do?

- Filter untrusted Input
- White-List vs. Black-List
- **a-zA-Z0-9\s** is often sufficient
- use Escape-Functions, like `mysql_real_string_escape()`
- avoid Functions that use the Shell, like `popen(3), system(3), glob()` (Perl), `<>` (Perl), etc.

What did I not tell you?

… a lot

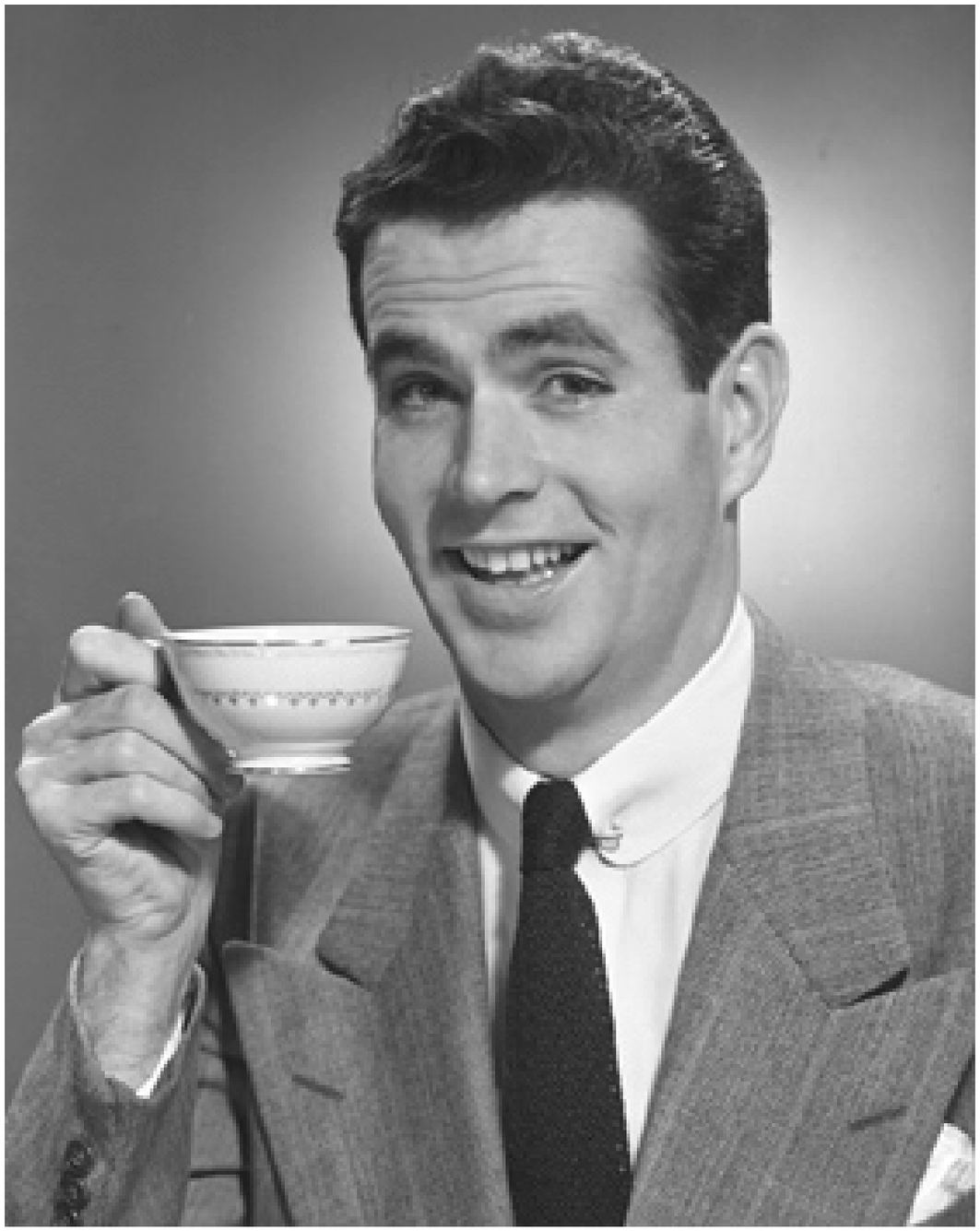
see Paper

„Sicherheitsrelevante Programmierfehler“

and Slides

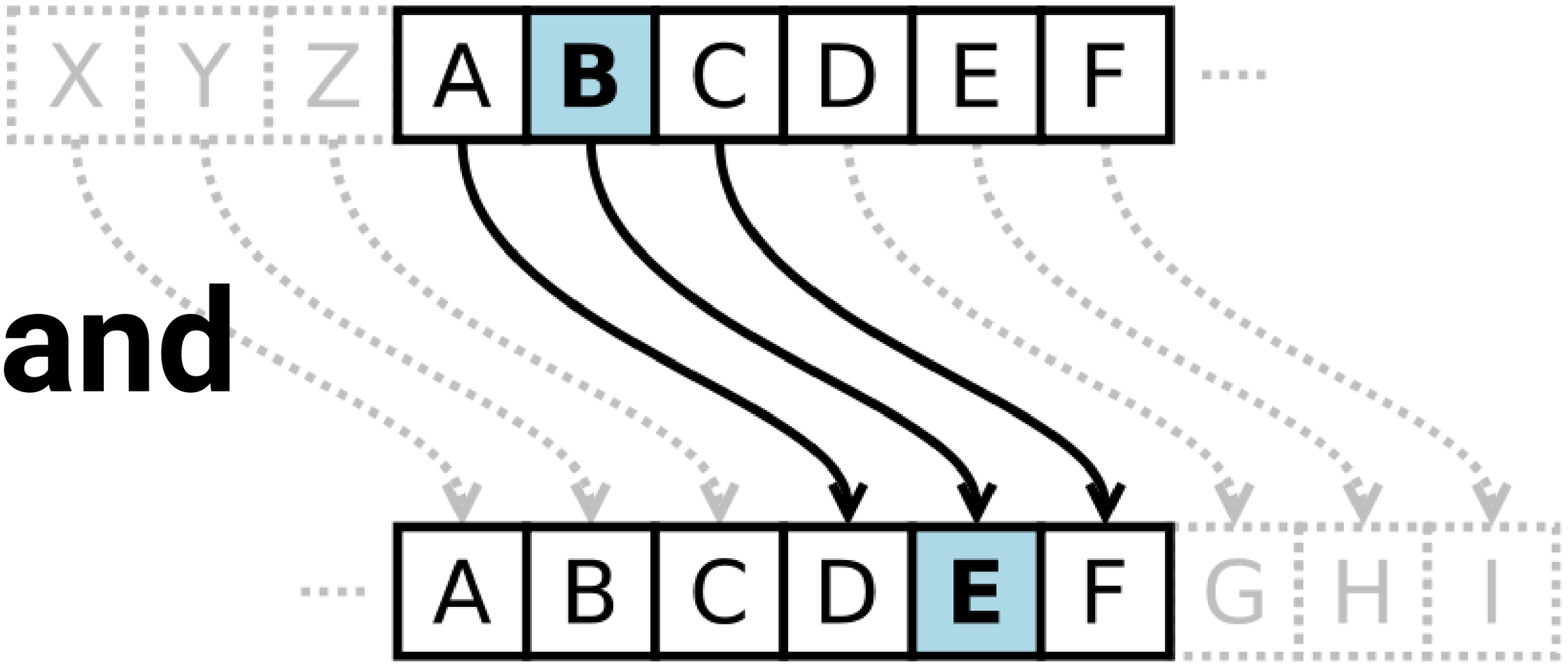
„Web-Security Basics“

10 min Coffee Break :-)



Cryptography

and



Randomness

Random Numbers

- max. Entropy: $H_{max} = 1$, if $P = 0{,}5 +/- e$; for $e = 0$
- **statistical** Randomness vs. **cryptographical** Randomness
- important Building Block for Algorithms and Protocols
- ... but hard to find on a deterministic Machine

Random Numbers

- most Unix-like Systems provide `/dev/random` or `/dev/urandom`
- `read(2)` instead of `fread(3)` (no Buffering/Read-Ahead)
- Usage of Hash Algorithms hide real Quality
- **avoid** PRNGs from Software Libraries
- if Kernel Support is not available, use EGADS or EGD
- or: *Counter-Mode Block-Cipher* and a random IV

novell-NLDAPsdk and BBS

- The Code uses a *Blum-Blum-Shub* (BBS) Generator as Fallback
- Unfortunately it generates the random Seed (which is the important Component regarding Security of this Algorithm) from System Time
- **System Time has only a few Bits of Entropy**
  - Result: Can be guessed relatively easy and therefore weakens the whole Crypto building on it.

novell-NLDAPsdk and BBS

- Lesson learned:
  - #1: Do not use the System Time as random Source
  - #2: Entropy is a valuable Resource. When you have *n* Bits of Entropy you will never get more from it with deterministic Operations, but it is possible to accidently reduce Entropy. (MD5 of 512 Bits of Entropy results in 128 Bits of Entropy)

CASA/novell-lum: Password Salt

- *Salt* is used to stop a *Code Book Attack*
- *novell-lum* used static *Salts*
- CASA used the Hash Value of the Password as *Salt* (therefore constant regarding Password)
  - Result: Password can be cracked easily.
- Lesson learned:
  - use a *random* and *changing Salt*

CASA: Password Cache

- *micasad* secures the cached Password by XOR'ing it with the Output of C#'s `Random()` Function seeded with a Timestamp
- Result: Two values XOR'ed with the same Value can be XOR'ed together to remove the "Key"
- Result: Assuming 12 hrs Runtime needs 2^39 Operations to decrypt **all** Values
- Result: `Random()` is predictable if an Attacker knows the Output (no *Forward Security*); Timestamp is BAD

CASA: Password Cache

- Lessons learned:
  - #1: In this Case using a Timestamp is ok but needs an additional Value which changes everytime (i.e. a *Sequence Number*)
  - # 2: Better add an additional Secret (i.e. a *Master Password*)

CASA: Password Store

- CASA encrypts the Password Store with AES in CBC Mode
- When calling `RijndaelManaged.CreateEncryptor()` the *Initialisation Vector* (IV) is not random/ changing
  - Result: Every Data encrypted under the same Key has the same IV which leaks Information about the first Plaintext Block

Secure Socket Layer

SSL and TLS

kopete and im.novell.com

- Problem #1: The first Version of *Kopete*'s GW Plugin does not verify the SSL-Cert :(
  - Result: Everyone can sniff/modify GWIM Traffic
- Problem #2: The SSL-Cert for `im.novell.com` was incorrect because of `CN=im.provo.novell.com` and an unknown CA-Cert
  - Result: *Kopete* opens a Accept/Reject Dialog

kopete and im.novell.com

- Problem #3: interactive Cert Verification is not possible because the Cert and its Fingerprint is not displayed
  - Result: Users start to click “Accept” to get their Work done
  - Result: All GWIM can be sniffed.
- Problem #4: All other GWIM Clients work without stumbling over the invalid Cert :(

kopete and im.novell.com

- Lessons learned:
  - #1: verify received Certs!
  - #2: only use valid Certs ;-)
  - #3: be as verbose as possible when using interactive Cert Verification
  - #4: make needed CA-Certs available for your Clients (safely)

Red Carpet Client and Python

- *rug* was written in *Python* and uses its `HTTPSConnection` Class
  - Result: Everyone can sniff RC Traffic
- Comment in *Python* Docmentation:

```
ssl(  sock[, keyfile, certfile])
Initiate a SSL connection over the socket sock. keyfile is the name
of a PEM formatted file that contains your private key. certfile is
a PEM formatted certificate chain file. On success, a new SSLObject
is returned.
Warning: This does not do any certificate verification!
```

Red Carpet Client and Python

- *rug* used *HTTP Digest Authentication* (HDA) encapsulated in SSL without Integrity Checking
- Result: After we broke the SSL Part we were able to read the Traffic in Plaintext. Without the Integrity Check from HDA we were also able to **modify** the command Stream!

Red Carpet Client and Python

- Lessons learned:
  - #1: RTFM ;-)
  - #2: A Chain of Security Mechanisms is only as strong as its weakest Link
  - #3: Use a secure SSL add-on for *Python*

novell-lum and User-friendliness

- The Components communicate with the LDAP Server via an SSL connection
- If the Component is not able to verify the Cert it uses the Callback Routine provided by *OpenSSL* to import a possibly missing CA-Cert - instead of using this Routine for interactive Verification (as designed) - and returns *SUCCESS*
- Result: Everyone can sniff/modify Traffic
- Result: Remote, un-authorized Logins

novell-lum and User-friendliness

- Lessons learned:
  - #1: RTFM ;-)
  - #2: Do not try to be clever by assuming an Operation Environment/Security Policy. And do not use questionable Workarounds based on this Assumtions.

Dos and Don’ts

- SSL Version >= 3 (TLS)
- take care, some Algorithms are out-dated (DES)
- take care, Keys can be too small
  - symmetric Ciphers: **Key-Length >= 128 Bit**
  - asymmetric Ciphers: **Key-Length >= 1024 Bit**
- take care, “exportable” Ciphers are weak and not needed
- take care, never loose the Private-Key

Dos and Don’ts

- use *Ephemeral Keying* = *Perfect Forward Secrecy* (PFS)
- PRNG needs a random Seed (at least as much random Bits as Length of Key to be generated)
- SSL-Session Parameters need to be re-newed when big Amounts of Data are transfered (Hash Collisions, Wrap-Arounds of Seq. Numbers, etc.)
- Directory of CA-Certs should only be write-able by *root* (take care with LDAP Directories and alike)
- and at least: **Verify the Certificates!**

What can I do?

- Am I safe with *AppArmor, SELinux, etc.?* No!
- Am I safe with all this nifty *gcc* Options? No!
- So, am I lost? No!
  - fully understand the Prg. Language you use
  - check your Program’s Input (Type, Size, ...)
  - use existing Crypto Libraries when needed
  - if in doubt, ask the Security-Team other Developers

http://www.suse.de/~thomas/papers/

“Sicherheitsrelevante Programmierfehler”

“Security Guidelines”

http://www.suse.de/~thomas/projects/

use *libsecprog*