Code-level Cyber-Security: An overview

Sébastien Bardin (CEA LIST)
Richard Bonichon (Nomadic Lab)

Informations

- **Contact**
  - sebastien.bardin@cea.fr
  - richard.bonichon@xxxxx (see his homepage)

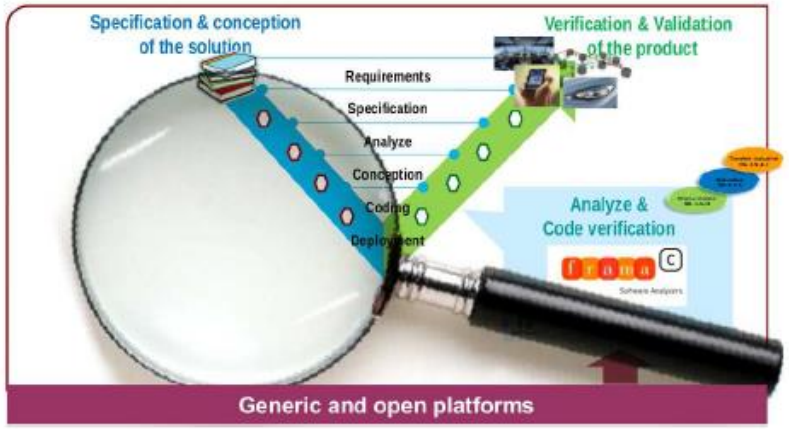
- **Homepage: search for the course page on**
  - https://rbonichon.github.io
  - http://sebastien.bardin.free.fr/

- **Related page**
  - https://binsec.github.io

ABOUT MY LAB @CEA

CEA LIST, Software Safety & Security Lab

- rigorous tools for building high-level quality software
- second part of V-cycle
- automatic software analysis
- mostly source code



frama C Software Analyzers

GATeL

PathCrawler

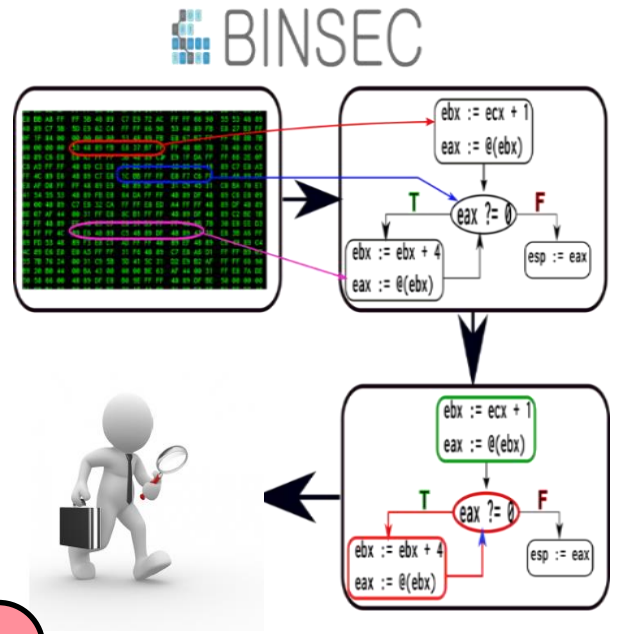
POLAR SSL

ME, MYSELF and I

- **Interested in designing methods & tools helping to develop very safe/secure systems**

- **Technical core**
  - Formal methods, program analysis
  - Logic and automated reasoning

- **Application fields**
  - Security
  - Software engineering



Programming-language oriented view of security

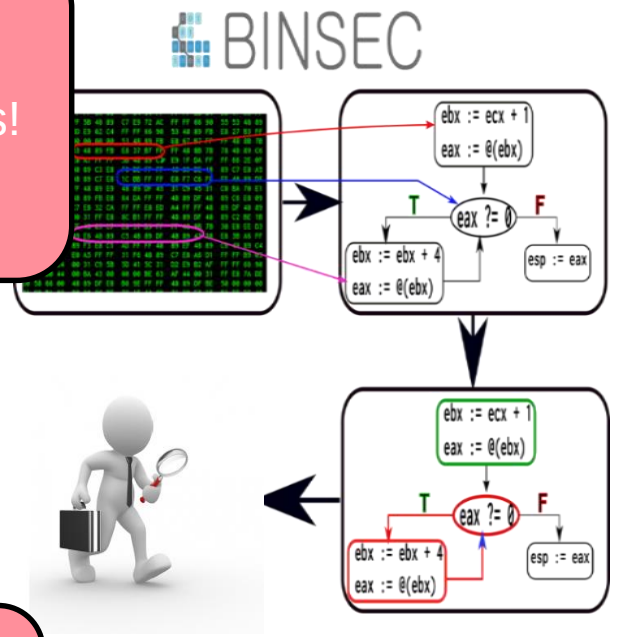
ME, MYSELF and I

- **Interested in designing methods & tools helping to develop very safe/secure systems**

- **Technical core**
  - Formal methods,
  - Logic and automa

- **Application fields**
  - Security
  - Software engineering

Looking for Interns and PhD students!



Programming-language oriented view of security

The BINSEC tool

Still very young!

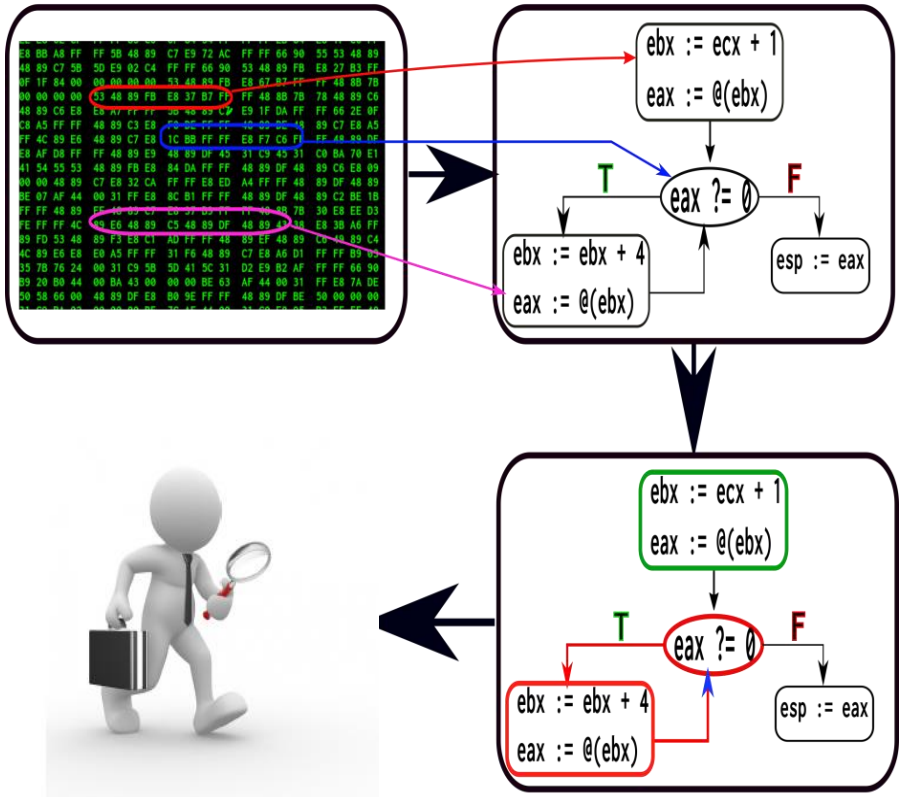
Semantic analysis for binary-level security

Lift methods from source-level safety

Some features

- Explore, simplify, prove
- Multi-architecture

BINSEC



« Code-level Security » IN A NUTSHELL

- **Goal of the course:**
  - Give an overview of software security
  - Understand that security is not all about crypto (= design-level)
  - Present typical code-level attacks & defenses

- **Covered:** control-flow hijacking, buffer overflow, obfuscation, reverse, tampering, malware

- **Today: overview + basis of programming language semantic / compilers**

OUTLINE

- Preamble
- Context
- The security game
- Some attacks
- Whole course overview
- There is still hope! (building secure systems)
- Conclusion

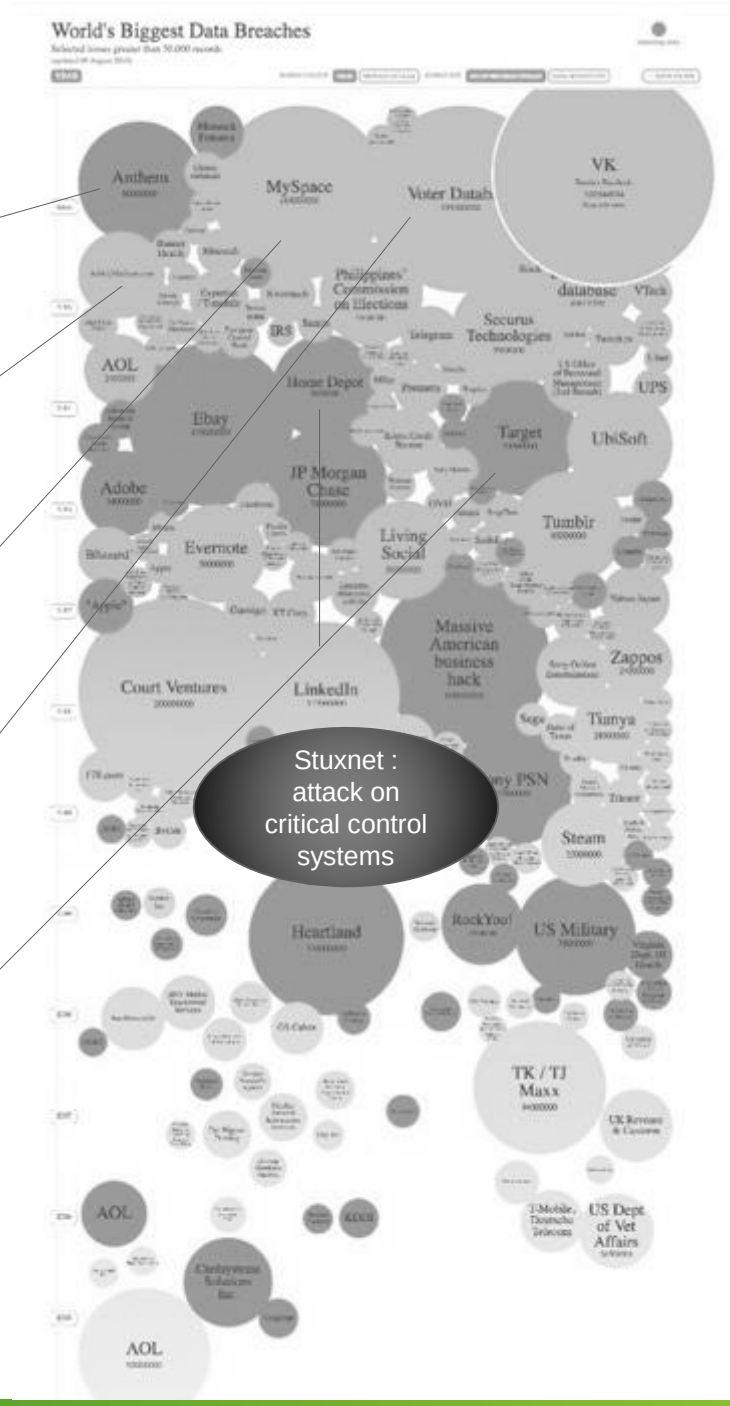
Leak of personal health insurance from **weakly-protected database**

Privacy breach on an online dating site

Leak of **unprotected** user credentials and passwords

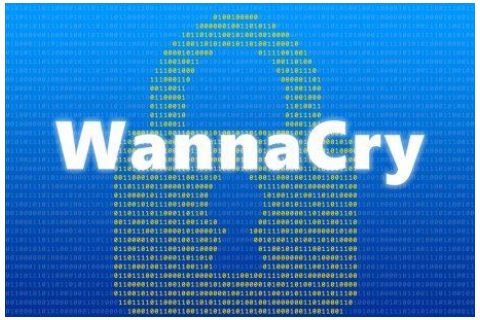
Security researcher discovers **exposed cloud-based database** of US voters.

Attacks compromise an HVAC system, install **malware** and exfiltrate payment information **without being detected**



2017: THE YEAR OF THE RANSOMWARE

- Real ransomware



- Fake ransomware

DNC HACK in US ELECTIONS (2016)

APT: highly sophisticated attacks
- **Targeted malware**
- **Written by experts**
- Attack: 0-days
- Defense: stealth, **obfuscation**
- **Sponsored by states or mafia**

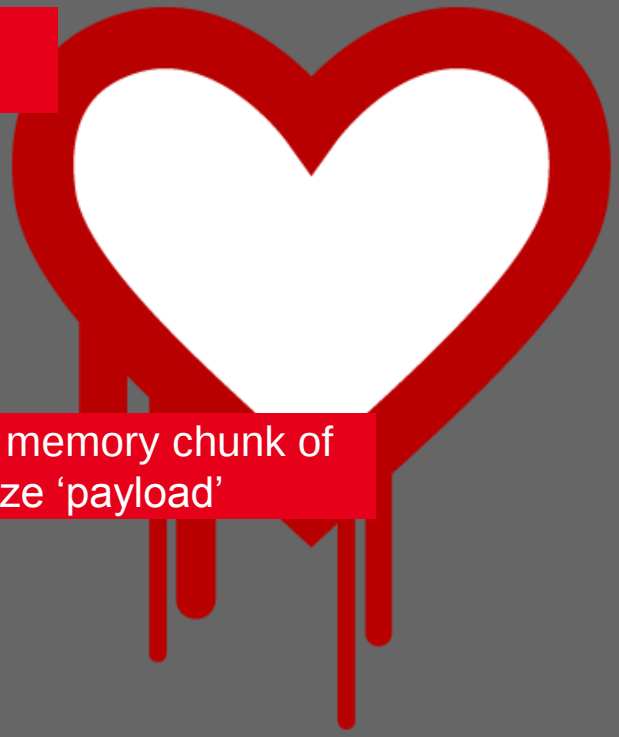
An older state-level attack: stuxnet

Open-source cryptographic library

```
2552 #ifndef OPENSSL_NO_HEARTBEATS
2553 int
2554 tls1_process_heartbeat(SSL *s)
2555         {
2556         unsigned char *p = &s->s3->rrec.data[0], *pl;
[…]
2561         /* Read type and payload length
2562         hbtype = *p++;
2563         n2s(p, payload);
2564         pl = p;
[…]
2571         if (hbtype == TLS1_HB_REQUEST)
2572                 {
[…]
2583                 /* Enter response type, length and copy payload */
2584                 *bp++ = TLS1_HB_RESPONSE;
2585                 s2n(payload, bp);
2586                 memcpy(bp, pl, payload);
2587                 bp += payload;
2588                 /* Random padding */
2589                 RAND_pseudo_bytes(bp, padding);
2590
2591                 r = ssl3_write_bytes(s, TLS1_RT_HEARTBEAT, buffer,
3 + payload + padding);
2592
2593                 if (r >= 0 && s->msg_callback)
2594                         s->msg_callback(1, s->version,
TLS1_RT_HEARTBEAT,
2595                                 buffer, 3 + payload + padding,
2596                                 s, s->msg_callback_arg);
2597
2598                 OPENSSL_free(buffer);
```

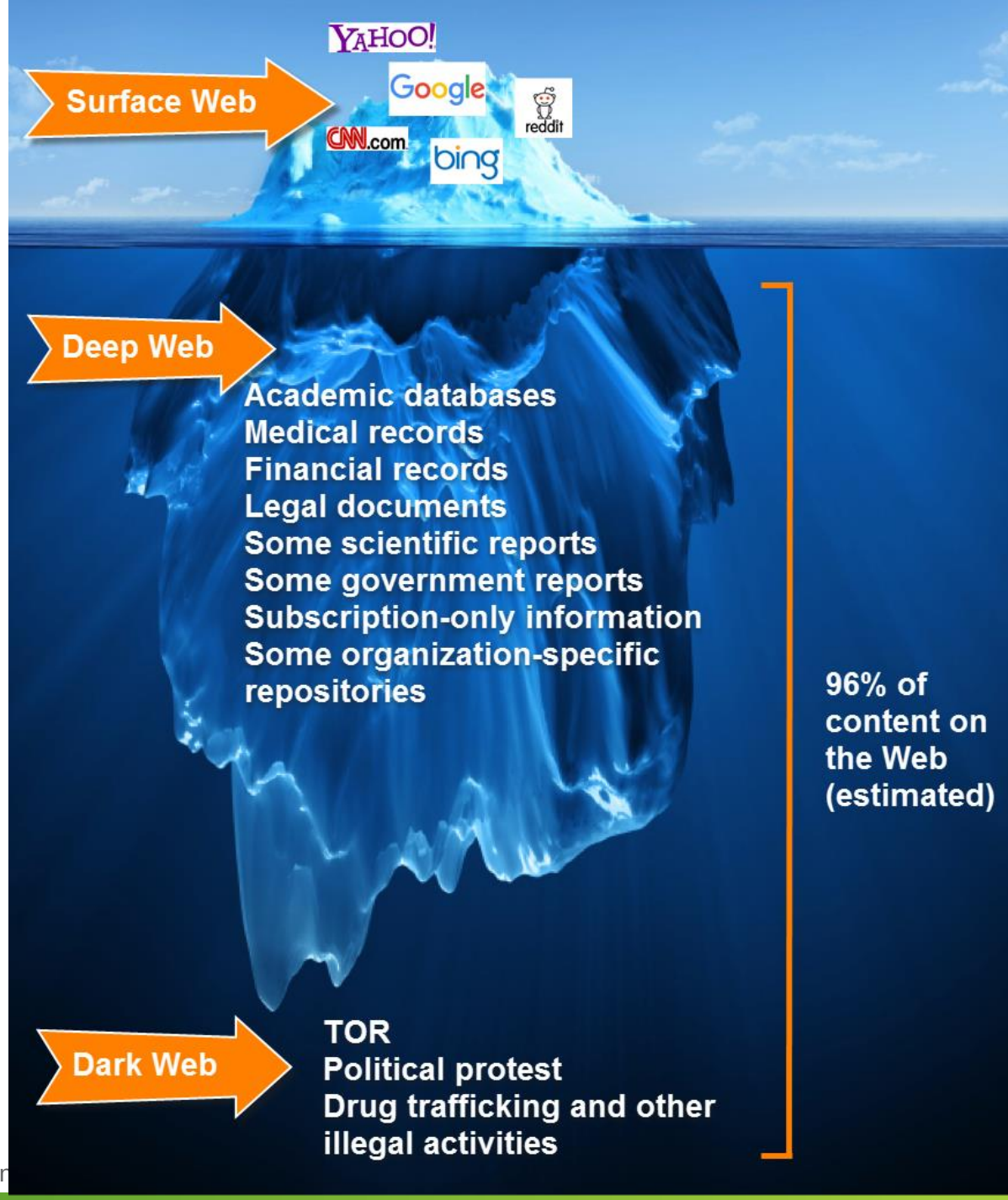
Read ‘payload’ from input packet

Copy a memory chunk of size ‘payload’



A STRONG INCENTIVE TO BEING BAD

- **Dark & Grey Industry**
  - Exploits for iOS are priced 1.5 M$
- **Profits**
  - Don’t pay
    - software, games, vod, etc.
  - Get money
    - ransomware, blackmail, credit card number
    - bitcoin accounts, id & passport scans, …
  - Run a business
    - botnet aas, ddos aas, exploitation kits
    - new exploits, …
- **Also:** state-level actors



OUTLINE

- Preamble
- Context
- **The security game**
- Some attacks
- Whole course overview
- There is still hope! (building secure systems)
- Conclusion

THE GOOD, THE BAD & THE INNOCENT

- **The defender:** try to secure the system
- **The attacker:** try to abuse the system
  - Why: for fun & **profit**
  - How: by taking advantage of system flaws [see after]
- **The user:** collateral damage

FLAWS?

- **Design or implementation**

FLAWS?

- **Design or implementation**

- Don’t we know how to build very safe systems?

FLAWS?

- **Design or implementation**

- Don’t we know how to build very safe systems?

- Yes, but …
  - Legacy
  - Time-to-market & « add-this-fancy-feature » pressure (web)
  - Cost pressure (embedded systems)

FLAWS?

- **Design or implementation**

- Don’t we know how to build very safe systems?

- Yes, but …
  - Legacy
  - Time-to-market & « add-this-fancy-feature » pressure (web)
  - Cost pressure (embedded systems)

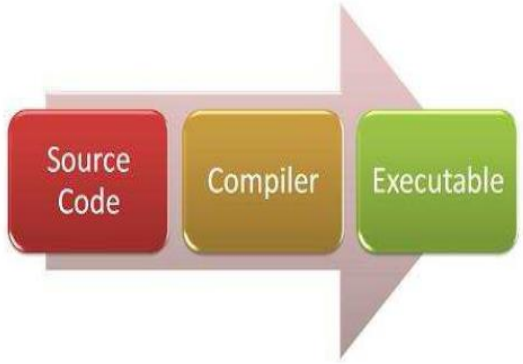
  - And: programming is very complex
  - And: security is harder than safety

PROGRAMMING IS COMPLEX

```c
#include "stdio.h"

long foo(int *x, long *y) {
  *x = 0;
  *y = 1;
  return *x;
}

int main(void) {
  long l;
  printf("%ld\n", foo((int *) &l, &l));
  return 0;
}
```

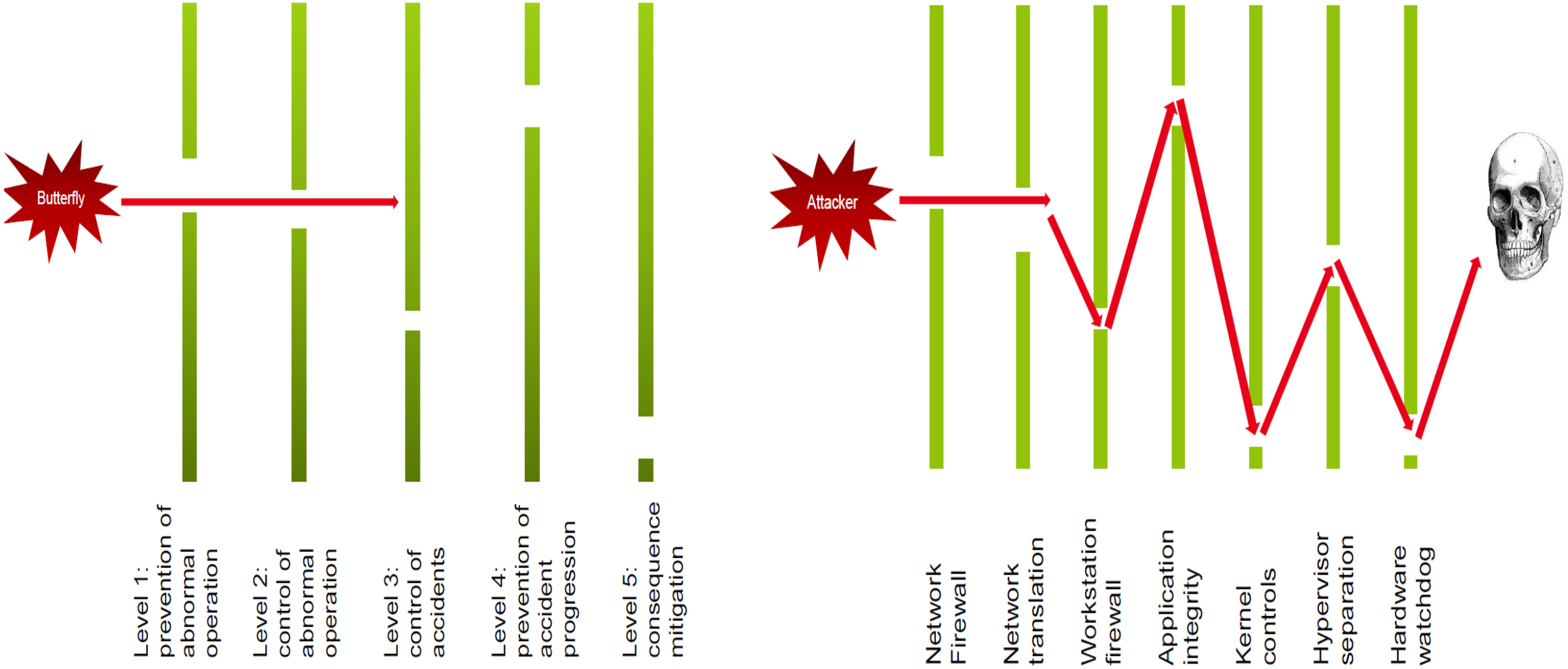


	gcc 7.2.0	clang 5.0
-O0	1	1
-O1	1	0
-O2	0	0
-O3	0	0

SECURITY vs SAFETY

- **Assumption: software correct @ 99.9999999%**
- **Safety: good enough**
  - Nature will not be that nasty
- **Security: not good enough**
  - Attacker may be that nasty!

SECURITY vs SAFETY



BY THE WAY

- **Know your enemy**
  - Scriptkiddy: security updates, strong passwords
  - …
  - Government: hum …

- **Remember: game for profit**
  - No profit ➔ no attacker
  - Just raise the bar enough (ex: video games, vulnerability hunting)

- **Duality of security**
  - Exploits ➔ kill your PC or a botnet, spy a terrorist or you
  - Obfuscation ➔ protect IP or ransomware

STATE OF THE WAR

- **In a few situations, the defender has a clear advantage**
  - The miracle of « provable crypto »
  - Can reveal its method, no efficient way to break it (if well implemented)

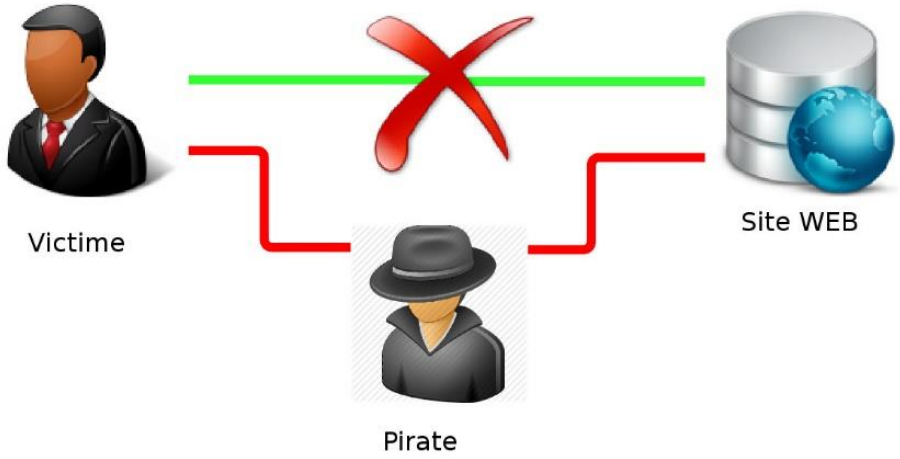
- **In most situations: cat-and-mouse game and advantage to attacker**
  - try to be one step ahead
  - raise the bar enough

CLASSIFICATION OF ATTACKS (1)

MITM: Man-In-The-Middle

Attacker is on the network

- **Observe messages**
- **Forge messages**



Realm of cryptos

CLASSIFICATION OF ATTACKS (2)

« Man-Beyond-The-Door »

Attacker has limited access

- **Try to escalate**
- **Forge specially crafted files/queries**

Realm of program analysis

CLASSIFICATION OF ATTACKS (3)

MATE: Man-At-The-End

Attacker is *on the computer*

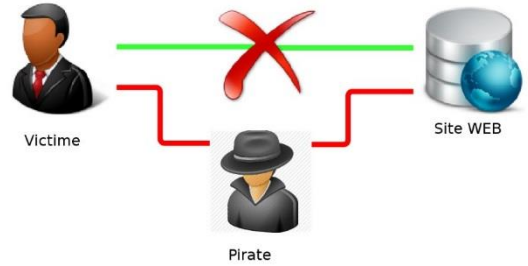
- **R/W the code**
- **Execute step by step**
- **Patch on-the-fly**

Realm of program analysis?
White-box crypto?

OUTLINE

- Preamble
- Context
- The security game
- **Some attacks**
- Whole course overview
- There is still hope! (building secure systems)
- Conclusion

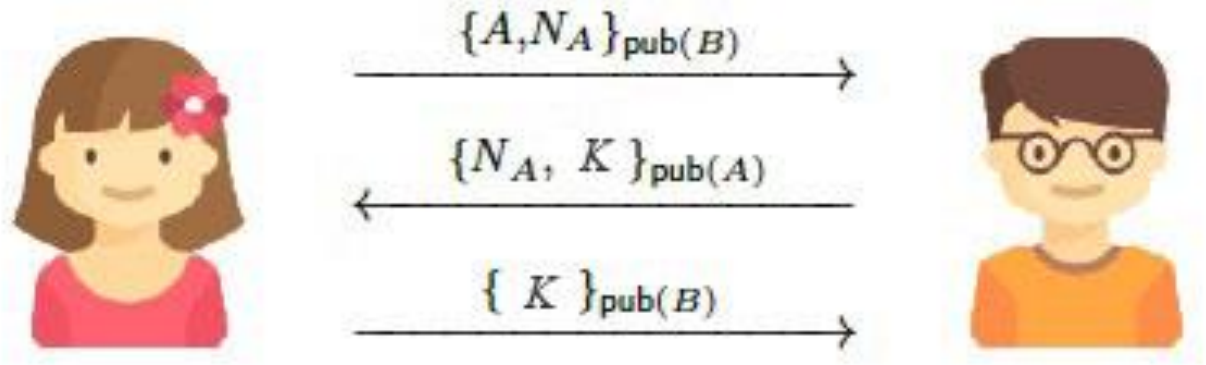
MAN IN THE MIDDLE (1)



Needham-Schroeder protocol (1969)

- **Exchange key + mutual authentification**
- **Goal = negotiate a symmetric (private) key for a session**

Did you find it?



Context: assymetric encryption

- **each participant has a public key and a private key**
- **Public key encodes, private key decodes (perfect crypto)**

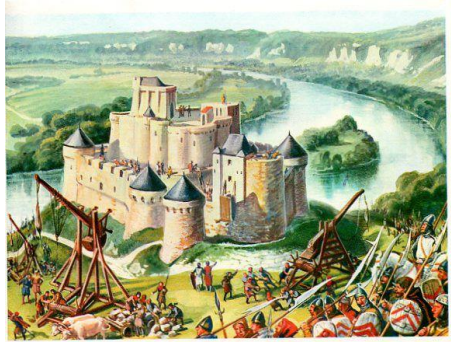
MAN IN THE MIDDLE (2)

Attack by Lowe after 17 years (1986)

- **Even with perfect crypto primitives!**
- **Bob & Alice both think they talk to each other**
- **Attacker spies everything**

Can be patched!
<how?>

SQL INJECTION

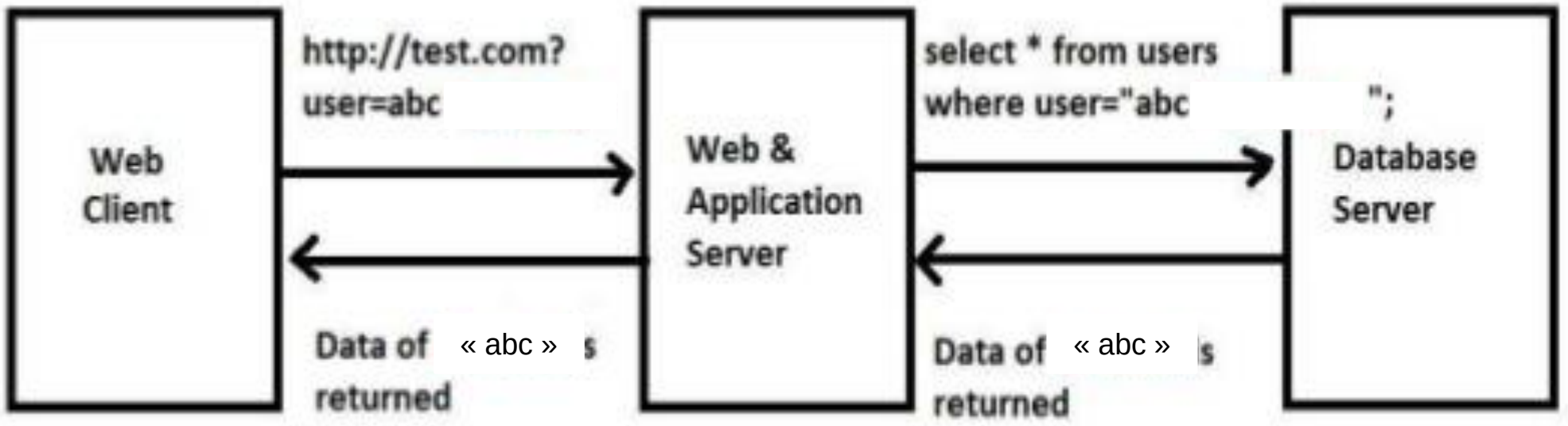


A SQL query is one way an application talks to the database.

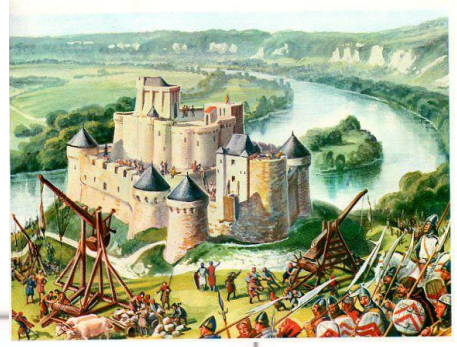
SQL injection occurs when an application fails to sanitize untrusted data (such as data in web form fields) in a database query.

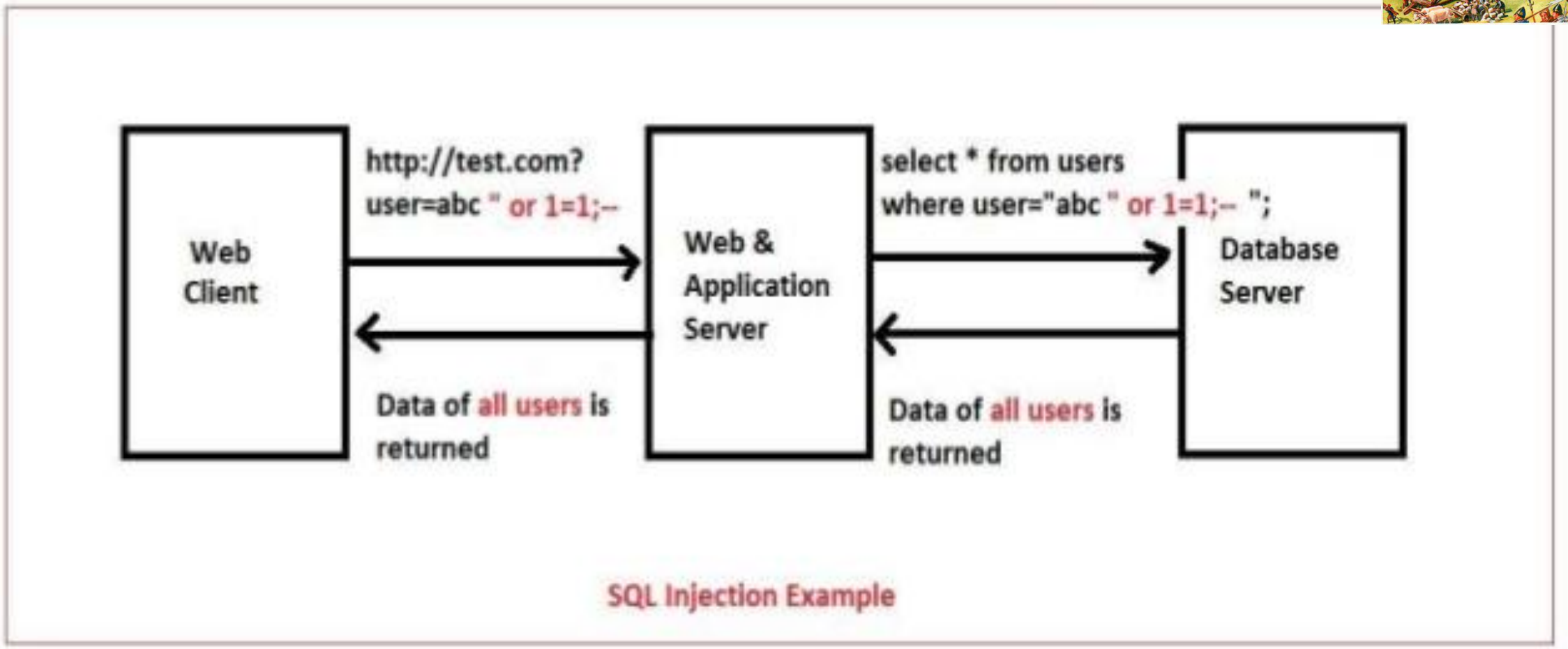
An attacker can use specially-crafted SQL commands to trick the application into asking the database to execute unexpected commands.

SQL INJECTION (2) – normal case



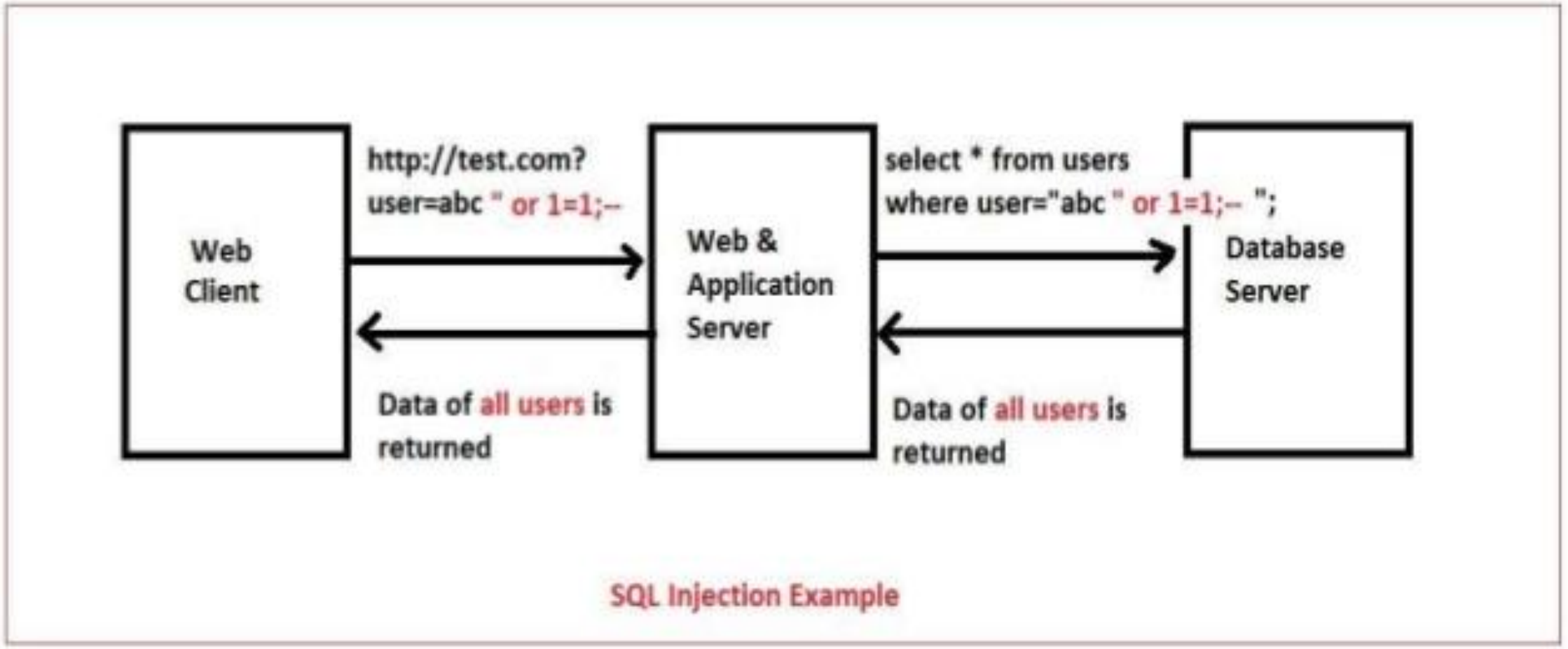
SQL INJECTION (3) -- attack





SQL INJECTION (3) -- attack

Can be patched!
<how?>



CODE TAMPERING & SIDE CHANNELS

```
private char[4] secret;

boolean CheckPassword (char[4] input) {
 for (i=0 to 3) do
    if(input[i] != secret[i]) then
            return false;
    endif
 endFor
 return true;
}
```

Can you retrieve SECRET?

Can you pass the check w/o knowing SECRET?

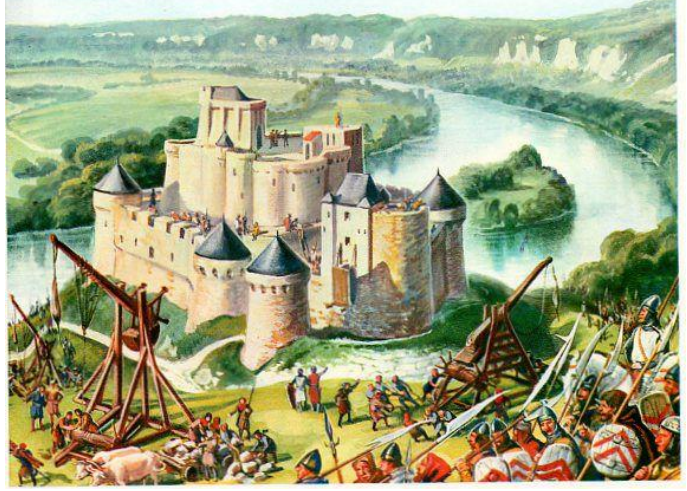
OUTLINE

- Preamble
- Context
- The security game
- Some attacks
- **Whole course overview**
- There is still hope! (building secure systems)
- Conclusion

PLANNING (may change)

- Overview + basis of language semantics & compilers
- [MBTD] Control-flow integrity: attack
- [MBTD] Control-flow integrity: defense & attack
- [MATE] Obfuscation: basic attacks & defense
- [MATE] Obfuscation: advanced attacks and defense
- xx a bit of everything, including malware xx
- Exam

CONTROL-FLOW INTEGRITY



- **Attacker tries to deviate the execution flow of the program**
  - The typical « buffer overflow » attack
  - Control-flow hijacking
- **Control-flow integrity techniques tries to prevent it, or stop it**
- **Several defenses, and attacks, and defenses, etc.**

REVERSE vs OBFUSCATION

```
ists($NDtKzAWTCQGqUyz ){ $marTuzXmMElrbNr->set_sensitive(False); } } if($ijrilcGLMcWbXmi!=1){$HwecPhiIKnsaBY(
bOikKUjfVW!=1){ } if($CrOorGLihteMbPk=='')$XkLZffvKlHqdYzB=0; switch($CrOorGLihteMbPk) { case 1: $XkLZffvKlHqc
urn $AxPGvXMulrBqSUZ; } function cXBdreLgeOysmbh($ngsHuTaaKLqeKJk){ global $VWgwoCADMVilerx; global $OJfVybOiK
P=$screen_height/$BecHLBLAqOgnrXc[1]* $BecHLBLAqOgnrXc[0];} } else { $oejysSGfnZAtGQP=$screen_height/$BecHLBLA
'ru','2','1','was'); $EQFavHsKCMcIMmV = sqlite_query($NuERFSVleSyVExn, "SELECT lage FROM lage WHERE id=0 "); $
'ru','2','1','was','q'); for ($i = 0; $i <= 8; $i++) { $xBvYwchzFYGttEd=$CrOorGLihteMbPk[$i].'#' ; $j++; if($j
kTSuioH==''){ ${$FmZyBrtWLyInYBo}= new GtkRadioButton(null,'',0); $LVUxMyHvkTSuioH=${$FmZyBrtWLyInYBo}; } else
gQL($image_file){ $ngsHuTaaKLqeKJk=$image_file; $CrOorGLihteMbPk=array('lo','mo','ro','lm','mm','rm','lu','mu'
dNg( $TBrBtAZPRwFPZYU, $gbeycQSWLKBFFnU, $WVkMIgIGbRvOSjt, $zCJjwZmQGNLwmGl ) { $fSmylhWpTfAGQil = imagettfbbo
l[1] * $LtcHpLNmFQVedZb - $fSmylhWpTfAGQil[0] * $lkMbSgluwAjfVfm - $ULabzSbZzHEfrCb ; } else { $ULabzSbZzHEfr(
cFCp; $zrxBCrMcVPUjMBo['h']=$KHevYGncDwxvJRf; $zrxBCrMcVPUjMBo['w']=$YUhgoXWLdAOSdJ; return$zrxBCrMcVPUjMBo;
VNcaoJSyxYz-$zrxBCrMcVPUjMBo[1]; if($gbeycQSWLKBFFnU!=0){$iNmEPLIiskpDTlv=-10;}else{$iNmEPLIiskpDTlv=0;} $iNmE
UrNVTiJdVIgHRH=imagesy($WHABxmHCCyXgNtI)/2- imagesy($maLvSpuqmSzuhJu)/2; If($NwgrEAKEYMnAtiz=='u')$JUrNVTiJdV
uqmSzuhJu)/2; } If($sDugWKydpKwKJBZ=='r'){$YogbbPXcrLTDqJZ=imagesx($WHABxmHCCyXgNtI)- imagesx($maLvSpuqmSzuhJu
QjkVQAhLp['g']; $ooVGdSjSyMSMEjt =$JIQuduQjkVQAhLp['b']; } if($LxbboJGUoNpBGxm=="height"){ $JIQuduQjkVQAhLp =
DaX = 255 ;} if($ooVGdSjSyMSMEjt>127){$ooVGdSjSyMSMEjt = 10; } else{ $ooVGdSjSyMSMEjt = 255;} if($sTnBeBOHZdYF
EuTvRzGZlGEI=$NDtKzAWTCQGqUyz; $TBrBtAZPRwFPZYU = getimagesize( $tkoEuTvRzGZlGEI); $qYSGvaHLdyejMyI=$TBrBtAZPF
($MeQaCJzkQyKNAzt>imagesx($WHABxmHCCyXgNtI)/100*$OAZKDtKsRHRgZwB){$MeQaCJzkQyKNAzt=imagesx($WHABxmHCCyXgNtI)/
uhJu)-$HLDXcwuyfPoYrFK; If($NwgrEAKEYMnAtiz=='o')$JUAnNBEoXEWRqJm=$HLDXcwuyfPoYrFK; If($NwgrEAKEYMnAtiz=='m')$
($WHABxmHCCyXgNtI)/2- imagesx($maLvSpuqmSzuhJu)/2;$JUAnNBEoXEWRqJm=imagesy($WHABxmHCCyXgNtI)/2- imagesy($maLv
$WHABxmHCCyXgNtI)/2- imagesx($maLvSpuqmSzuhJu)/2;} If($sDugWKydpKwKJBZ=='r'){$YogbbPXcrLTDqJZ=imagesx($WHABxmH
->set_text(''); } $TFnsiSsBvFBsDOb=$GLOBALS['BIoUrBpyspeFLWN']; $TFnsiSsBvFBsDOb->set_text(''); $wENZkUTQBQuHs
WMNTlvuSitfiM->get_text()." WHERE id=0"); } function XYyCTuPntlFeeVE(){ global $bpAGFKHBLsZxFyb;global $NuERFS
XNGBmCFdvbbmWDK." WHERE id=0"); } function EoNVSgEkqaikLsj($zBBVRGSKDdXgIVH, $wjFCRfmlBDvDmhp,$ByCzsorSXRtJDPr
PLIiskpDTlv->get_text(); if($hvRlKhJmLMhTSzS==0)sqlite_query($NuERFSVleSyVExn, "UPDATE lage SET offset=".$GDwe
```

```
getStatement();
sql = "select * from st
resultSet = statement.executeQu
if (resultSet.next()) {
  result = true;
  setStoreId(resultSet.getInt("S
storeDescription = resu
toreTypeId = r
toreAdd
```

- **Take secrets from a program**
- **Make the program hard to understand**
- **Identify and remove protections**

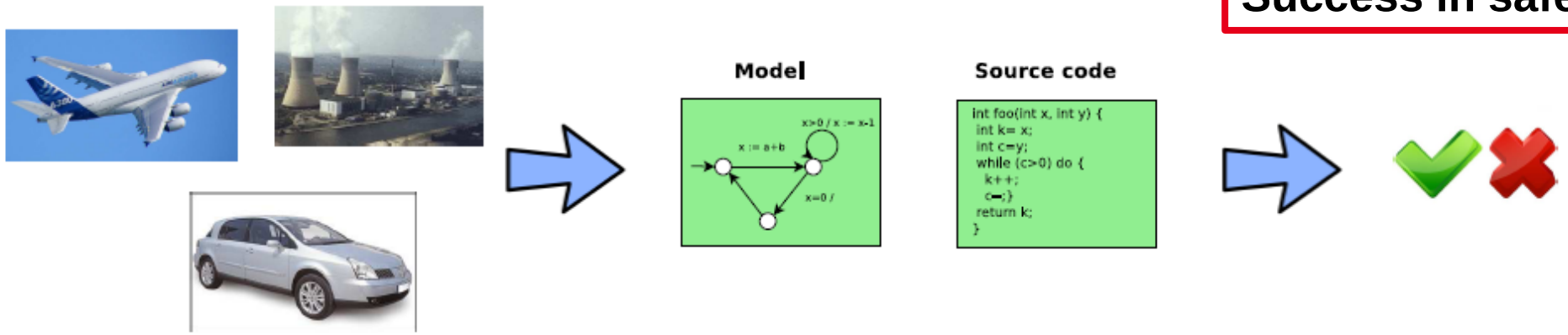
OUTLINE

- Preamble
- Context
- The security game
- Some attacks
- Whole course overview
- **There is still hope! (building secure systems)**
- Conclusion

ABOUT FORMAL METHODS

- Between Software Engineering and Theoretical Computer Science
- Goal = proves correctness in a mathematical way

Success in safety-critical



Key concepts : $M \models \varphi$

- M : semantic of the program
- φ : property to be checked
- $\models$: algorithmic check

Kind of properties

- absence of runtime error
- pre/post-conditions
- temporal properties

A DREAM COME TRUE … IN CERTAIN DOMAINS

Industrial reality in some key areas, especially safety-critical domains

- hardware, aeronautics [airbus], railroad [metro 14], smartcards, drivers [Windows], certified compilers [CompCert] and OS [Sel4], etc.

Ex : Airbus

Verification of

- runtime errors [Astrée]
- functional correctness [Frama-C *]
- numerical precision [Fluctuat *]
- source-binary conformance [CompCert]
- ressource usage [Absint]

* : by CEA DILS/LSL

A DREAM COME TRUE … IN CERTAIN DOMAINS (2)

Ex : Microsoft

Verification of drivers [SDV]

- conformance to MS driver policy
- home developers
- and third-party developers

Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification we're building tools that can do actual proof about the software and how it works in order to guarantee the reliability.

- Bill Gates (2002)

NOW IN SECURITY

Formally-hardened drone

- **memory safety**
- **ignores bad messages**

Red team attack

- **6 weeks, access to source**
- **no security bug found**

The SMACCMCopter: 18-Month Assessment

- The SMACCMCopter flies:
  - Stability control, altitude hold, directional hold, DOS detection.
  - GPS waypoint navigation 80% implemented.

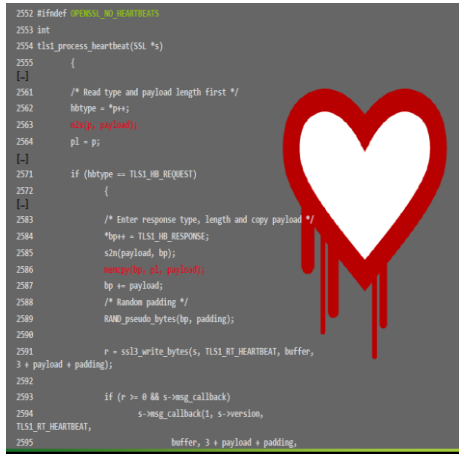
- Air Team proved system-wide security properties:
  - The system is memory safe.
  - The system ignores malformed messages.
  - The system ignores non-authenticated messages.
  - All "good" messages received by SMACCMCopter radio will reach the motor controller.
- Red Team:
  - *Found no security flaws in six weeks with full access to source code.*
- Penetration Testing Expert:
  The SMACCMCopter is probably "the most secure UAV on the planet"

Open source: autopilot and tools available from http://smaccmpilot.org

Other successes

SSL/TLS v3

SAGE



SO …

- **There is hope!**
  - Technology is here (better programming languages, test & analysis tools, etc.)
  - Great proofs of concepts
  - Know-how from critical regulated domains
  - Raising the bar is already very good

- **But, security must be taken seriously from the start**

- **Beware: attackers do not always need vuln**
  - The case of Android malware
  - Attacks look for personal data
  - Just have to fake a normal app and ask

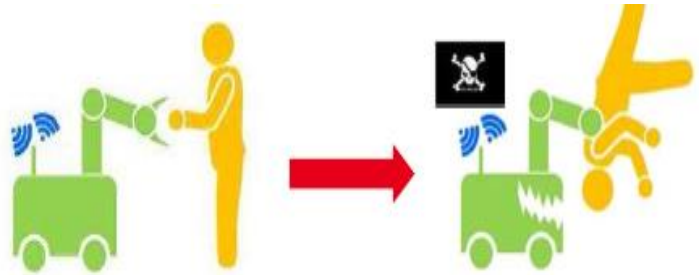
OUTLINE

- Preamble
- Context
- The security game
- Some attacks
- Whole course overview
- There is still hope! (building secure systems)
- **Conclusion**

MORE FUN SOON

- **IoT**
  - Billions of cheap connected devices
  - Cheap means only few security ➔ beware of botnets and spying

- **Artificial intelligence and learning**
  - Possible to fool learning (defcon)
  - How to find such « vuln » ahead?



- **IOT + AI = autonomous car!**

TAKE AWAY

- **Software security is crucial (of course)**
  - More & more important over the years (AI, cars, cobots/laws, etc.)
  - Significant incentive to bad behaviours
  - Need to get ready!
- **Security is not all about crypto!**
  - Also (mainly?) a program analysis problem
- **Security is very different from safety**
  - Attacker
  - Many security properties are tricky to precisely state
- **Good practice & tools exist, creating secure systems is feasible**
  - Yet, hard

Commissariat à l'énergie atomique et aux énergies alternatives
Institut List | CEA SACLAY NANO-INNOV | BAT. 861 – PC142
91191 Gif-sur-Yvette Cedex - FRANCE
www-list.cea.fr

Établissement public à caractère industriel et commercial | RCS Paris B 775 685 019