ENIGMA

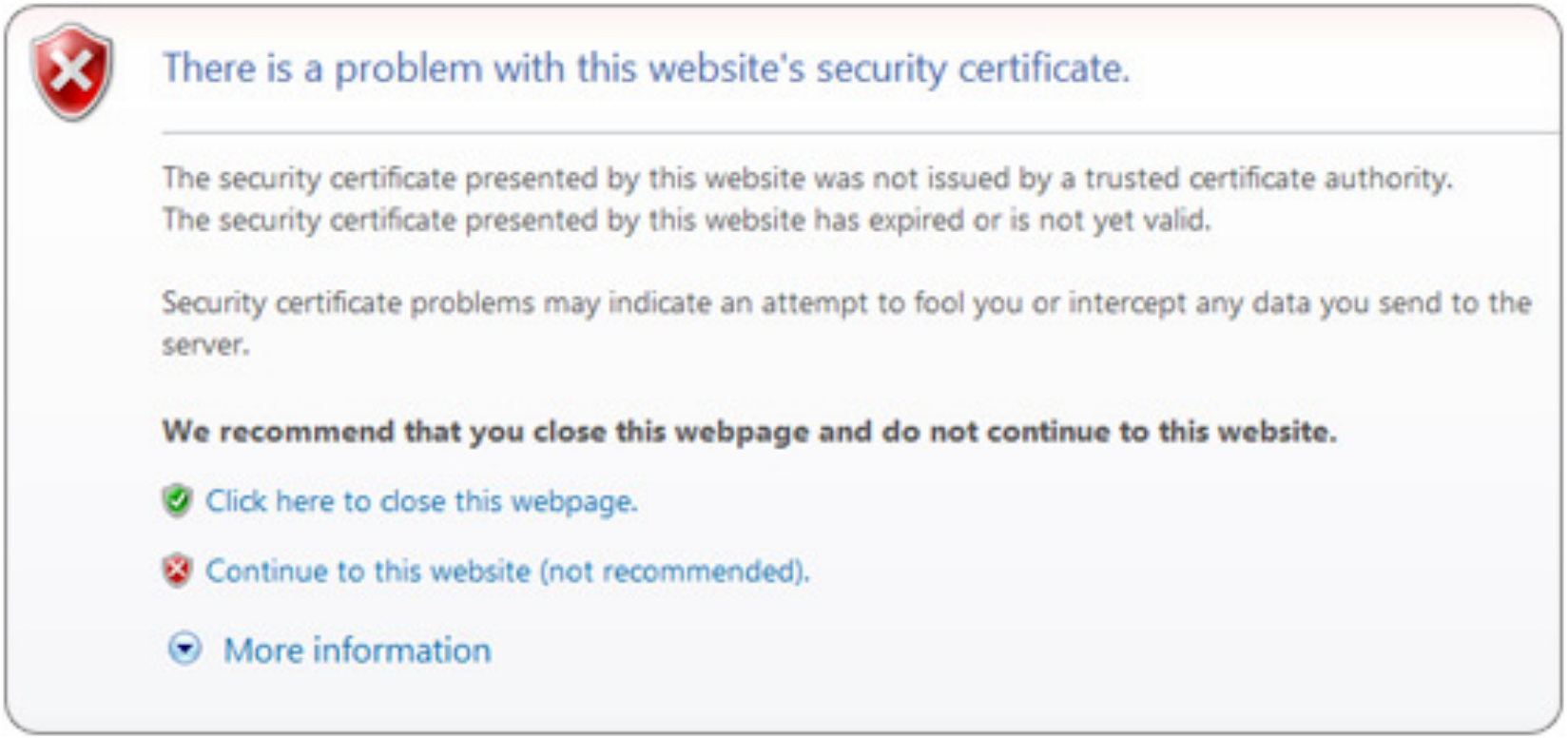
Usable Security
- The Source Awakens -

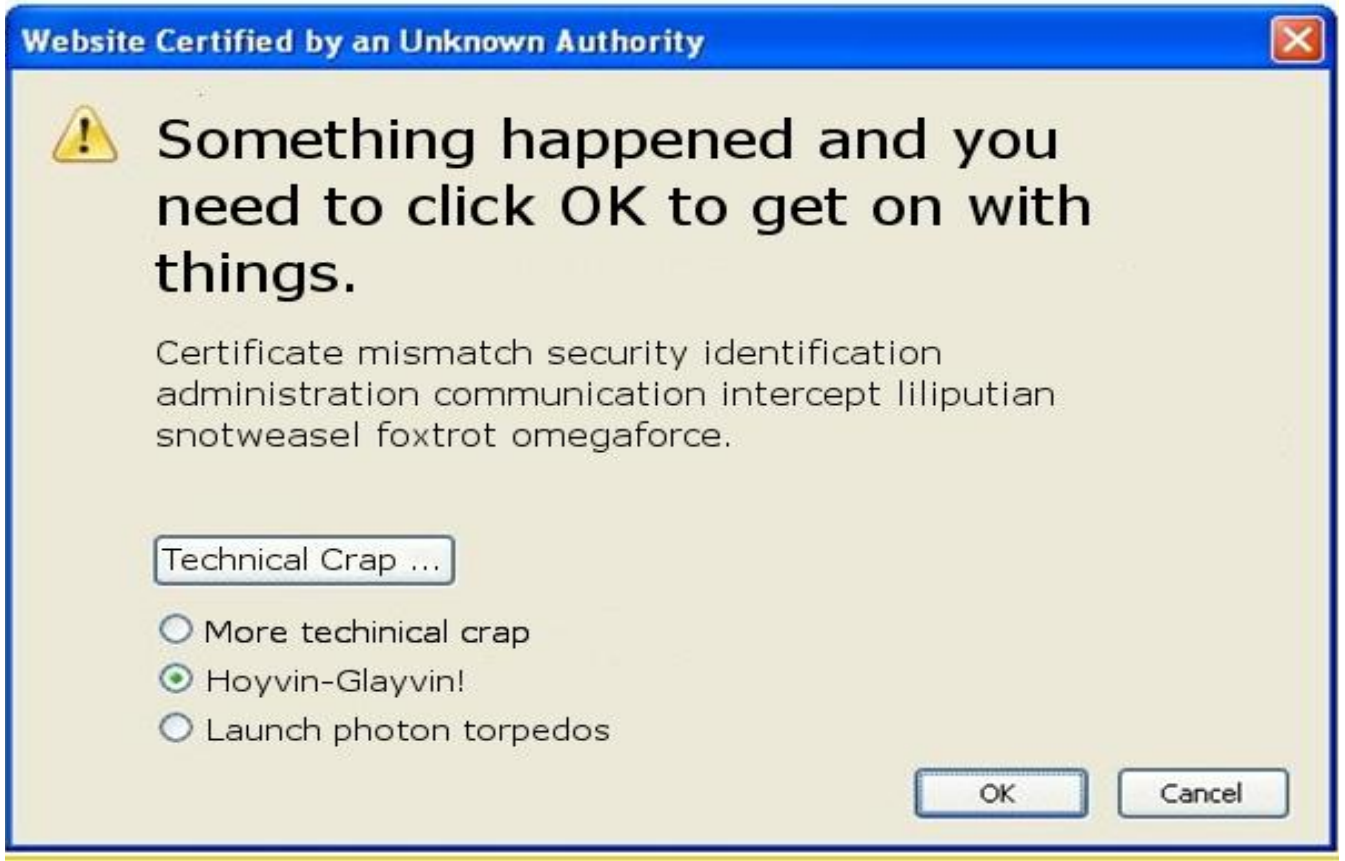
Matthew Smith – University of Bonn

“Users Are Not the Enemy”

Angela Sasse ’99

“Developers Are Not the Enemy Either”


There is a problem with this website's security certificate.
The security certificate presented by this website was not issued by a trusted certificate authority.
The security certificate presented by this website has expired or is not yet valid.
Security certificate problems may indicate an attempt to fool you or intercept any data you send to the server.
We recommend that you close this webpage and do not continue to this website.
Click here to close this webpage.
Continue to this website (not recommended).
More information



Adapted from Jonathan Nightingale

4.5M unique certificates in 2013

610k “bad” certificates

123456

correct horse battery staple

https://xkcd.com/936/

ZDNet SEARCH HARDWARE WINDOWS 10 IOT INNOVATION MOBILITY MORE NEWSLETTERS ALL WRITERS

MUST READ IF A SMARTPHONE VENDOR ACQUIESCES TO ANTI-ENCRYPTION LAWS, DON'T USE THEM

6.46 million LinkedIn passwords leaked online

Forbes / Security

Top 20 Stocks

NEWS

RockYou hack accounts

OCT 28, 2015 @ 10:10 AM 37,737 VIEWS

13 Million Passwords Appear To Have Leaked From This Free Web Host - UPDATED

Thomas Fox-Brewster, FORBES STAFF

I cover crime, privacy and security in digital and physical forms.

FOLLOW ON FORBES (162)

RISK ASSESSMENT / SECURITY & HACKTIVISM

13 million plaintext passwords belonging to webhost users leaked online

llion passwords were

Personal data is exposed as a result of a five-month-old hack on 000Webhost.

ecurity engineering Joseph Bonneau

stackoverflow.com

```
// these sizes are relatively arbitrary
int seedBytes = 20;
int hashBytes = 20;

// increase iterations as high as your performance can tolerate
// since this increases computational cost of password guessing
// which should help security
int iterations = 1000;

// to save a new password:

SecureRandom rng = new SecureRandom();
byte[] salt = rng.generateSeed(seedBytes);

Pkcs5S2ParametersGenerator kdf = new Pkcs5S2ParametersGenerator();
kdf.init(passwordToSave.getBytes("UTF-8"), salt, iterations);

byte[] hash =
    ((KeyParameter) kdf.generateDerivedMacParameters(8*hashBytes)).getKey();

// now save salt and hash

// to check a password, given the known previous salt and hash:

kdf = new Pkcs5S2ParametersGenerator();
kdf.init(passwordToCheck.getBytes("UTF-8"), salt, iterations);

byte[] hashToCheck =
    ((KeyParameter) kdf.generateDerivedMacParameters(8*hashBytes)).getKey();
```

```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
                                 uint8_t *signature, UInt16 signatureLen)
{
    OSStatus        err;
    ...

    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;
    ...

fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
}
```


End Users
Administrators
and Developers

Trust me! I’m an engineer!

Story 1
HTTPS

The default Android HTTPS API implements correct certificate validation.

Q: I am getting an error of „javax.net.ssl.SSLException: Not trusted server certificate“.

[...]

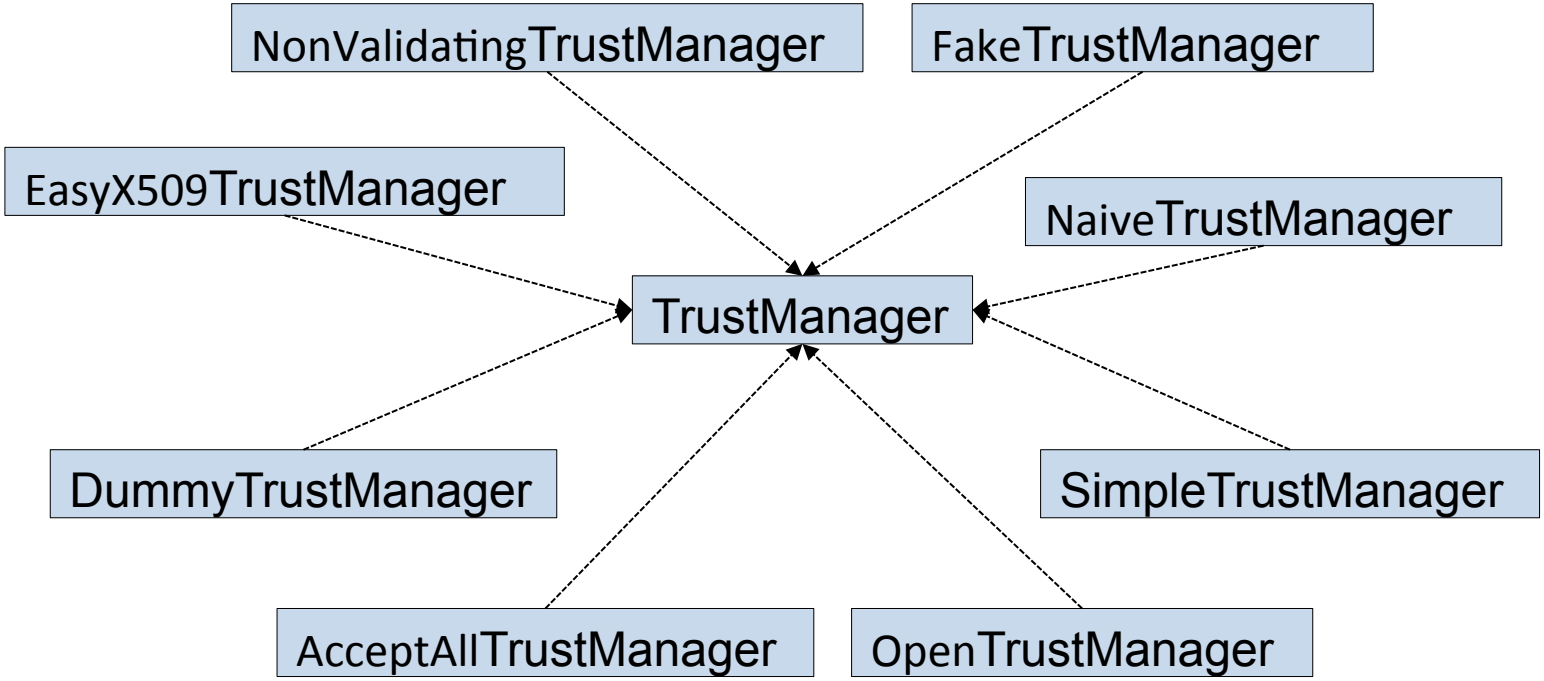
I have spent 40 hours researching and trying to figure out a workaround for this issue.

A: Look at this tutorial
http://blog.antoine.li/.../android-trusting-ssl-certificates

stackoverflow.com

```
// Create a trust manager that does not validate certificate chains
TrustManager[] trustAllCerts = new TrustManager[] { new X509TrustManager() {

        public java.security.cert.X509Certificate[] getAcceptedIssuers() {
                return null;
        }

        public void checkClientTrusted(X509Certificate[] chain, String authType) throws CertificateException {
                // do nothing
        }

        public void checkServerTrusted(X509Certificate[] chain, String authType) throws CertificateException {
                // do nothing
        }

} };
```

Static analysis of 13.500 popular Android Apps found thousands of vulnerable Apps



- Cherry-picked 100 apps
  - 21 apps trust all certificates
  - 20 apps accept all hostnames

These 41 Apps had an install base between 39 – 185 million devices!

- Captured credentials for:
  - American Express, Diners Club, Paypal, bank accounts, Facebook, Twitter, Google, Yahoo, Microsoft Live ID, Box, WordPress, remote control servers, arbitrary email accounts, and IBM Sametime, among others.

Problem → Solution

USEC Methods

Evaluating Without Users

Cognitive Walkthrough

Heuristic Evaluation

Model-Based Evaluation

Evaluating With Users

Qualitative

Silent Observation

Think Aloud

Constructive Interaction

Retrospective Testing

Interviews

Quantitative

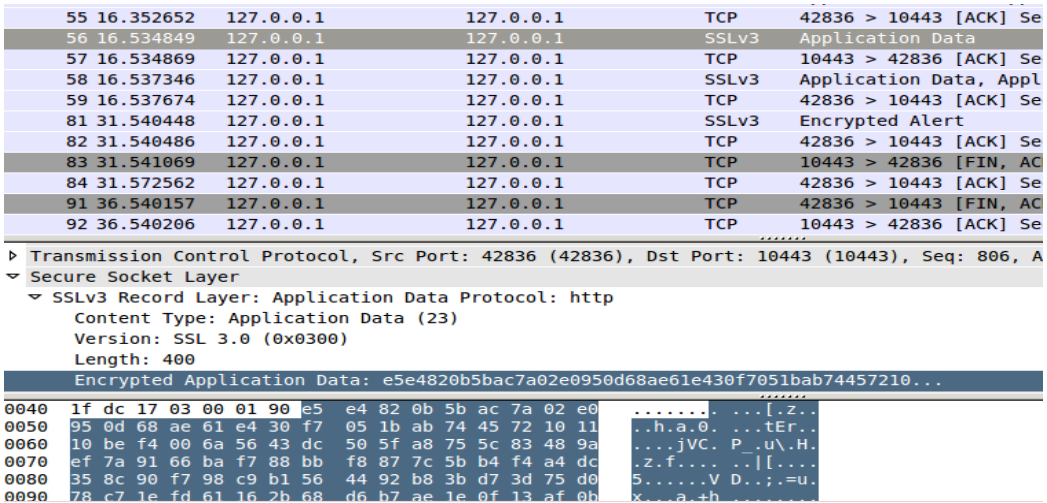
Controlled Experiments

Questionnaires

“This app was one of our first mobile apps and when we noticed that there were problems with the SSL certificate, we just implemented the first working solution we found on the Internet.”

“We use self-signed certificates for testing purposes and the easiest way to make them working is to remove certificate validation. Somehow we must have forgotten to remove that code again when we released our app.“

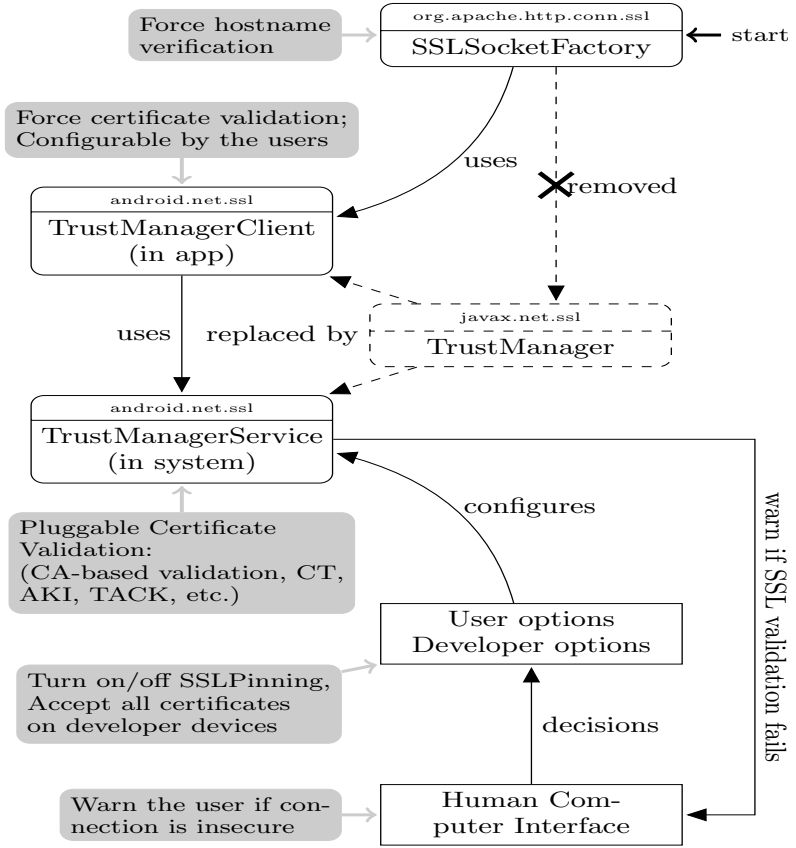
“[...] When I used Wireshark to look at the traffic, Wireshark said that this is a proper SSL protected data stream and I could not see any cleartext information when I manually inspected the packets. So I really cannot see what the problem is here.”



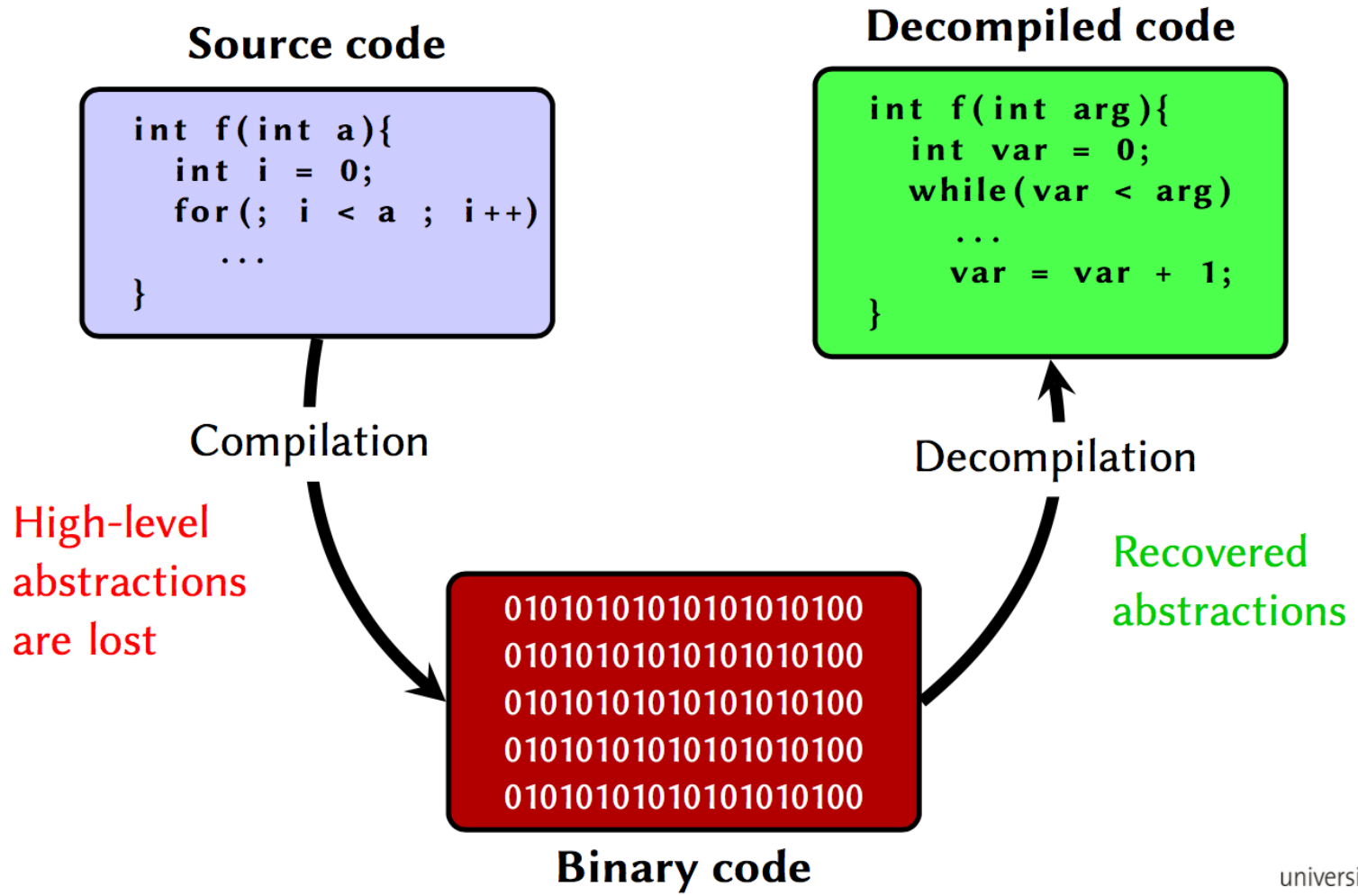
“The app accepts all SSL certificates because some users wanted to connect to their blogs with self-signed certs and […] because Android does not provide an easy-to-use SSL certificate warning message, it was a lot easier to simply accept all self-signed certificates.”

vs.

- ✔ HTTPS can be secure on Android
- ✔ Backwards compatible for 13.500 apps
  except
  - ✘ 19 apps that implemented pinning
  - ✔ updating those to the new pinning system would be very easy

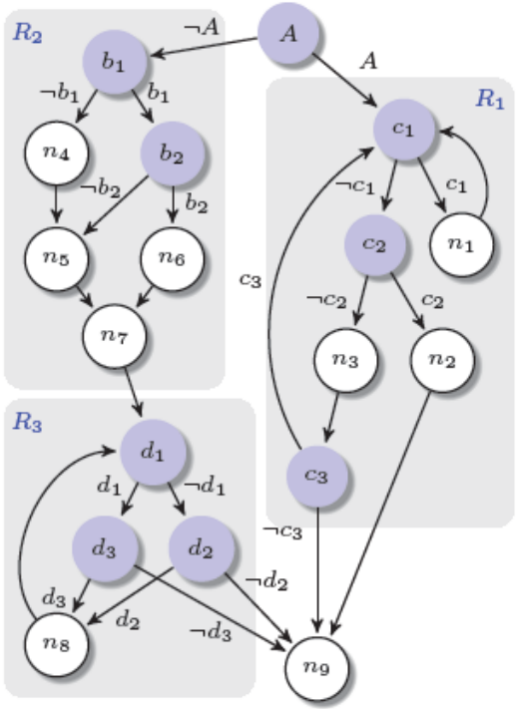


Story 2
Malware Analysis


Source code
int f(int a){
int i = 0;
for(; i < a ; i++)
...
}
Decompiled code
int f(int arg){
int var = 0;
while(var < arg)
...
var = var + 1;
}
Compilation
Decompilation
High-level abstractions are lost
Recovered abstractions
0101010101010101010
0101010101010101010
0101010101010101010
0101010101010101010
0101010101010101010
Binary code

P2P Zeus Sample

1,571 goto statements in 50k LoC




```
if (A)
  while (1)
    while (c1)
      n1
    if (c2)
      break
  n3
  if (¬c3)
    goto LABEL_4
n2
else
  if (¬b1)
    n4
    goto LABEL_1
  if (¬b2)
LABEL_1:
    n5
    goto LABEL_2
  n6
LABEL_2:
  n7
  while (d1)
    if (¬d3)
      goto LABEL_4
LABEL_3:
    n8
  if (d2)
    goto LABEL_3
LABEL_4:
n9
```

R1 R2 R3

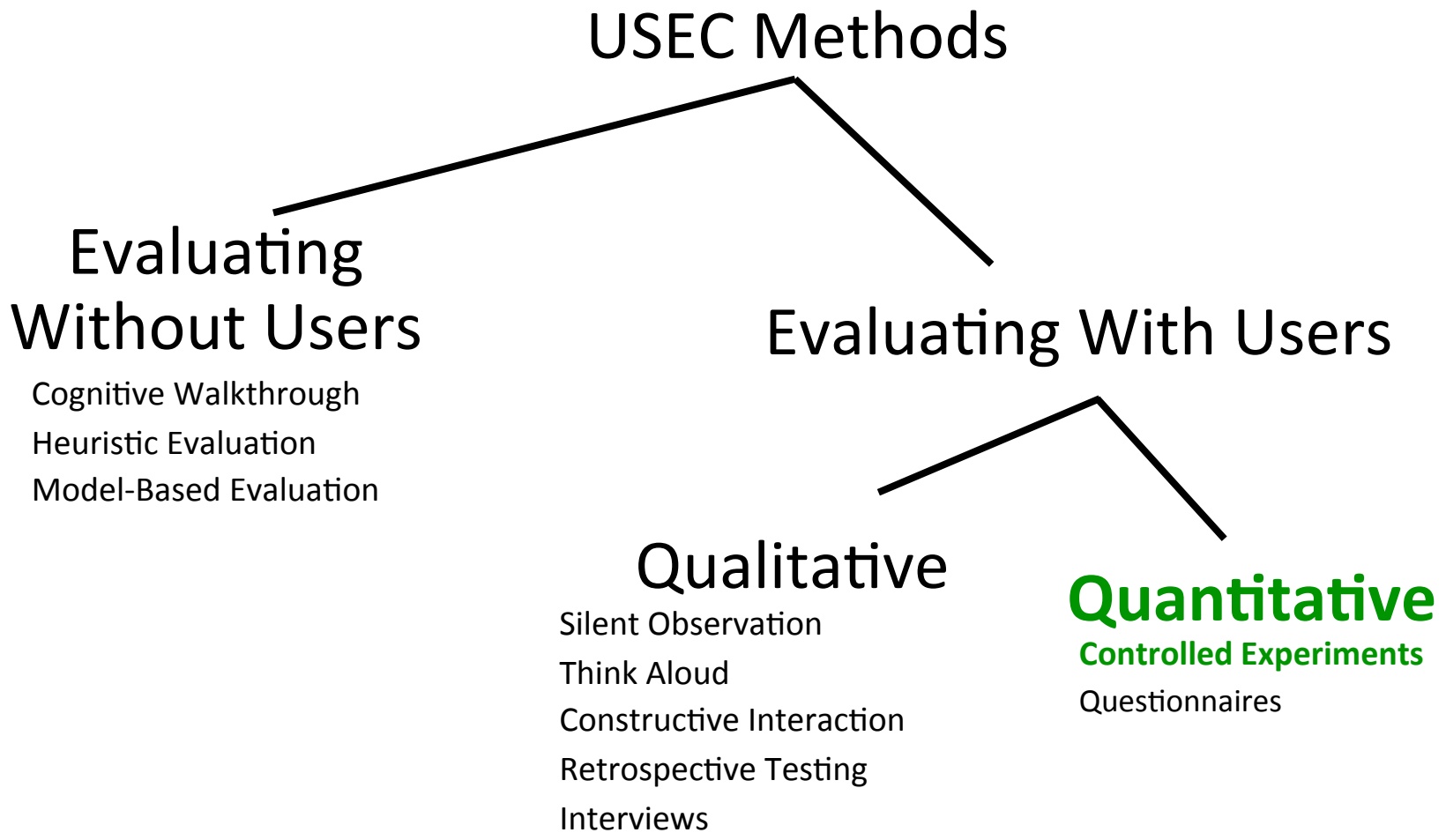
```
1  void *__cdecl sub_10006390(){
2    __int32 v13; // eax@14
3    int v14; // esi@15
4    unsigned int v15; // ecx@15
5    int v16; // edx@16
6    char *v17; // edi@18
7    bool v18; // zf@18
8    unsigned int v19; // edx@18
9    char v20; // dl@21
10   char v23; // [sp+0h] [bp-338h]@1
11   int v30; // [sp+30Ch] [bp-2Ch]@1
12   __int32 v36; // [sp+324h] [bp-14h]@14
13   int v37; // [sp+328h] [bp-10h]@1
14   int i; // [sp+330h] [bp-8h]@1
15   // [...]
16   v30 = *"qwrtpsdfghjklzxcvbnm";
17   v32 = *"ghjklzxcvbnm";
18   v33 = *"lzxcvbnm";
19   v31 = *"psdfghjklzxcvbnm";
20   v35 = aQwrtpsdfghjklz[20];
21   v37 = *"eyuioa";
22   v34 = *"vbnm";
23   v38 = *"oa";
24   v39 = aEyuioa[6];
25   // [...]
```

```
26     v14 = 0;
27     v15 = 3;
28     if ( v13 > 0 )
29     {
30       v16 = 1 - &v23;
31       for ( i = 1 - &v23; ; v16 = i )
32       {
33         v17 = &v23 + v14;
34         v19 = (&v23 + v14 + v16) & 0x80000001;
35         v18 = v19 == 0;
36         if ( (v19 & 0x80000000) != 0 )
37           v18 = ((v19 - 1) | 0xFFFFFFFE) == -1;
38         v20 = v18 ? *(&v37 + dwSeed / v15 % 6)
39                   : *(&v30 + dwSeed / v15 % 0x14);
40   ++v14;
41   v15 += 2;
42   *v17 = v20;
43   if ( v14 >= v36 )
44   break; }
45   }
```

Hex-Rays: Simda malware - Domain generation algorithm

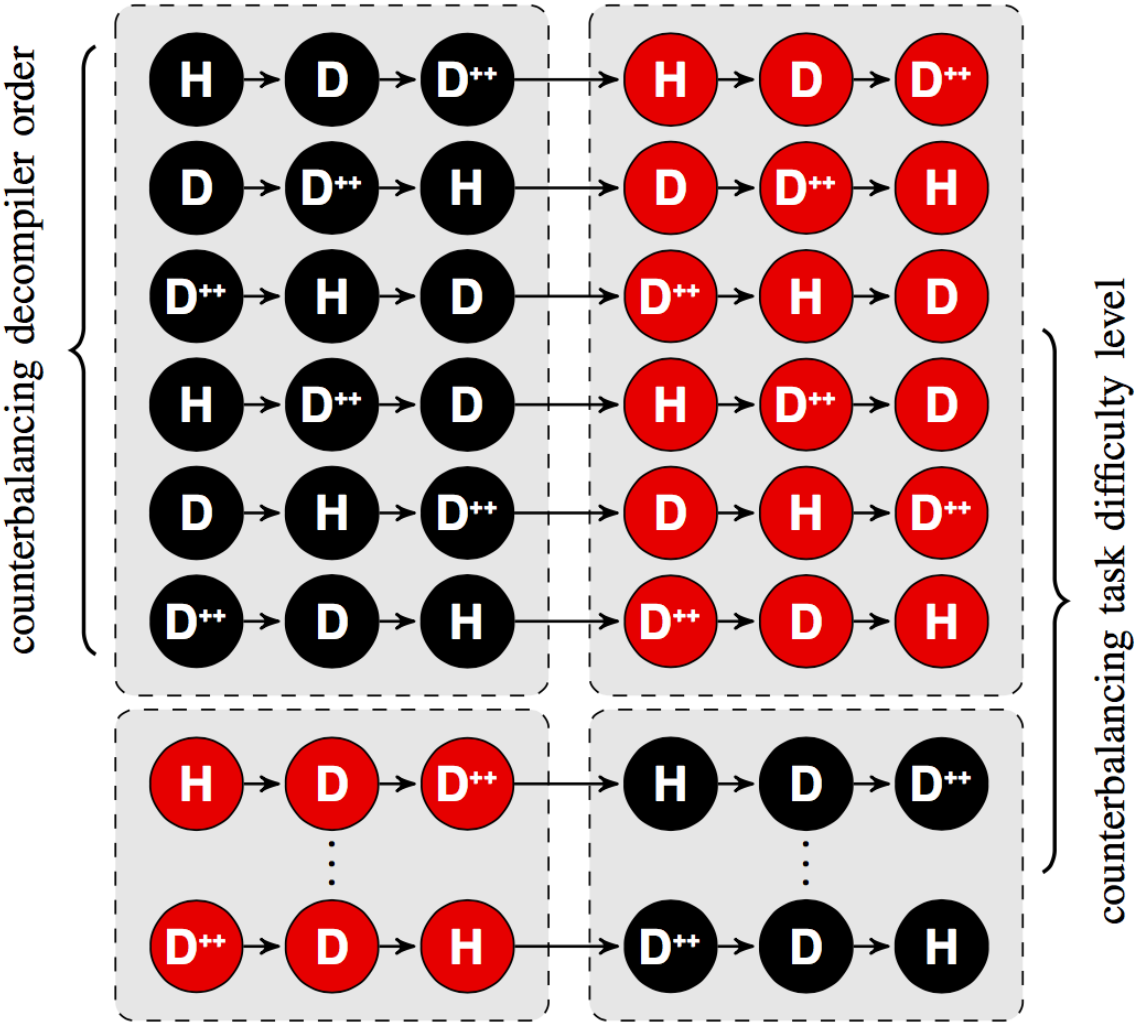
```
LPVOID sub_10006390(){
  char * v1 = "qwrtpsdfghjklzxcvbnm";
  char * v2 = "eyuioa";
  // [...]
  int v13 = 3;
  for(int i = 0; i < num; i++){
    char v14 = i % 2 == 0 ? v1[(dwSeed / v13) % 20]
                          : v2[(dwSeed / v13) % 6];
v13 += 2;
v3[i] = v14;
}
```

DREAM++ Simda malware - Domain generation algorithm


USEC Methods
Evaluating Without Users
Cognitive Walkthrough
Heuristic Evaluation
Model-Based Evaluation
Evaluating With Users
Qualitative
Silent Observation
Think Aloud
Constructive Interaction
Retrospective Testing
Interviews
Quantitative
Controlled Experiments
Questionnaires

Malware Analysis Study

- 3 Decompilers
  - HexRays
  - DREAM
  - DREAM++
- 6 Analysis Tasks
- 21 Students
- 9 Analysts



“The code mostly looks like a straightforward C translation of machine code; besides a general sense about what is going on, I think I'd rather just see the assembly.” - DREAM

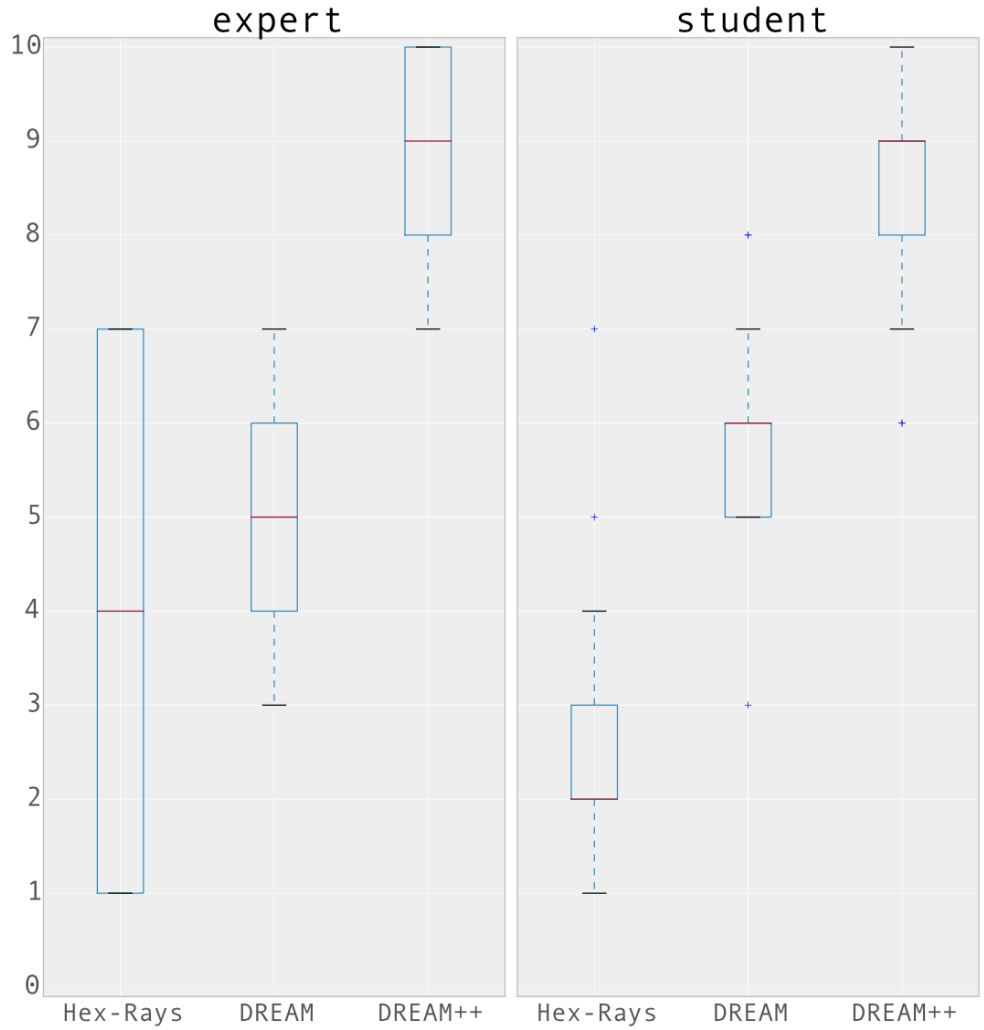
“This code looks like it was written by a human, even if many of the variable names are quite generic. But just the named index variable makes the code much easier to read! ” – DREAM++

Students

- Solved 3 times as many tasks with DREAM++ than with Hex-Rays

Experts

- Solved 1.5 times as many task with DREAM++ than with Hex-Rays



“Developers Are Not the Enemy”