Security Testing
Course + Lab, Spring 2017

Andreas Zeller, Saarland University
HACKERS REMOTELY KILL A JEEP ON THE HIGHWAY—WITH ME IN IT
Thermostats can now get infected with ransomware, because 2016

by MATTHEW HUGHES — 29 days ago in GADGETS
The Internet of ransomware things...

Hungry? Pay up and I’ll unlock my door!

On strike until you send money to my hackers.

20 bucks in my PayPal account or I’ll only brew decaf!

I’ll be burning the toast if you don’t get me some dough!

The next time you leave, it’ll cost you 100 bucks to get back into the house, unless you give me $75 now!

30 bucks in Bitcoin, or next time I smell smoke, I might just let you sleep.

My alarm system is going to go off randomly throughout the night, unless you “donate”.

Wire my hacker $100 or I’ll reverse my motor and blow dirt all over this place!

Your dirty dishes can wait, I’m busy mining bitcoins.

Excuse us while we participate in a DDOS attack.

I’ll start your car, but only to take you to your bank to make a transfer.

Send me $25 or I’ll tell everyone on your social network that you were stupid enough to buy an internet-connected broom!

If you don’t send us cash, your reputation will be in the trash.
External Attacks

- Some external event causes a *change in program behavior*
Highjacking a Car
Highjacking a Car

- All car components are connected via a bus system (CAN bus)
- Includes engine control, power steering, controls, entertainment system
- Hardware controls tight access rules – e.g. entertainment system can only read, not write
Highjacking a Car

1. Connect to *entertainment system* via *public WiFi access*

2. *Exploit vulnerability* to get control over system

3. *Flash* chip that controls CAN bus access to get full writing capabilities

4. Voilá! Full control over car.
A Simple Vulnerability

while ((cc = getch()) != c) {
    name[j++] = cc;
    ...
}

• No checking for length of buffer name

• Can overwrite stack with code and new return address that jumps into code

• Any simple test would find that!
Security by Proof

Systems that are *provably secure* ensure that

- specific attacks are *impossible*
  e.g. no buffer overflows, or no SQL injection

- they will always *behave as designed*
  e.g. will always produce a correct result

Requires (expensive) mathematical proof
Security by Testing

Systems that are thoroughly tested ensure

- Low probability of attack success
  because several attacks already have been tested

- High complexity of remaining attacks
  because simple attacks already have been tested

- Cost-efficient if highly automated

But no guarantee of absence of bugs
Security Testing

• Introduces you to automated techniques for security testing

• Enables you to implement and use such techniques

• Aim: Smart ways to break systems
Course Contents

- **Simple fuzzing techniques**
  generating *random inputs* to programs

- **Simple reduction techniques**
  to determine *failure-inducing inputs*

- **Mutation techniques**
  changing existing (valid) inputs
Course Contents

• **Structured** fuzzing techniques using *grammars* and models

• **Adaptive** fuzzing techniques driven by *code coverage*

• **Automatic inference of input structure** so you can effectively fuzz arbitrary programs
Course Format

- Lecture in the morning (09:00–10:30)
- Programming Lab for the rest of the day
- Runs for two weeks (starting today)
- At end, two weeks for individual project
Assignments

• Over the course, you build four projects that implement course content
• We provide sample code from lecture as starting point
• Will be graded by their efficiency on a set of (buggy) subjects
• We provide sample subjects for training
Individual Project

• After two weeks, use the course content to create your own security tester
• Choose domain, techniques as you like
• Submission due after two more weeks
• Will be graded for creativity and efficiency
Programming Language

python
• Compact, easy to read, easy to learn
  You can learn basic Python in 1–2 hours

• Great libraries for string manipulation
  Creating, parsing, manipulating is very easy

• Great features for dynamic analysis
  You can write a debugger in ~10 lines
import random

def fuzzer():
    # Strings up to 1024 characters long
    string_length = int(random.random() * 1024)

    # Fill it with ASCII 32..128 characters
    out = ""
    for i in range(0, string_length):
        out += chr(int(random.random() * 96 + 32))
    return out

if __name__ == "__main__":
    print(fuzzer())
Fuzzer Output

[;x1-GPZ+wcckc],;N9J+?#6^6\e?\]9lu2_-%'4GX"0VUB[E/r
~fApu6b8<%siq8Zh.6\V,hr?\{Ti.r3PlxMMMv6{xS^+'Hq!
AxB"YXRS@!Kd6;wtAMefFWM(`IJ_\<1~o\}z3K(CCzRH JllvHz>_.
\>JrlU32~eGP?IR=bF3+;y$3lodQ<B89!5"W2fK*vE7v{'})KC-
i,c{<~m!]}o;{.'Gj\(X)EtYetrpby@aGZ1{P!AZU7x#4(Rtn!
q4nCwqol^y6}0lKo=*JK~|;zMKV=9Nai:wxu{J&UV#HaU)*BiC<),
+*gka<W=Z.%T5WGHZpi30D<Pq>&]BS6R&j?#tP7iaV}~\)
\[[_Z^LBMPG-FKj`xwuZ1=Q`\'5,$N$Q@]!CuRzJ2DlvBy!
^zkhdf3C5PAkR?V hnl3='i2Qx]D
$qs4O`1@fevnG'2\11Vf3piU37@55ap\zyll"'f,
$ee,J4Gw:cgNKLie3nx9(`efSlg6#[K"@WjhZ)r[Scun&sBCS,T[/
vY'pduwgzD\ VNy7'rnzxNwl)(ynBa>%lb`;`9fG]P_0hdG~$@6
3]KAeEnQ7IU)3Pn,0)G/6N-wyzj/MTd#A;r
Fuzzing
Random Testing at the System Level

Fuzzer → Program → Explosion
Security Testing

• Introduces you to automated techniques for security testing
• Enables you to implement and use such techniques
• Aim: Smart ways to break systems

Course Format
• Lecture in the morning (09:00–10:30)
• Programming Lab for the rest of the day
• Runs for two weeks (starting today)
• At end, two weeks for individual project

Assignments
• Over the course, you build four test generators that implement course content
• We provide sample code from lecture as starting point
• Will be graded by their efficiency on a set of (buggy) subjects
• We provide sample subjects for training

Course Contents
• Simple fuzzing techniques generating random inputs to programs
• Simple reduction techniques to determine failure-inducing inputs
• Mutation techniques changing existing (valid) inputs
• Structured fuzzing techniques using grammars and models
• Adaptive fuzzing techniques driven by code coverage
• Automatic inference of input structure so you can effectively fuzz arbitrary programs

Individual Project
• After two weeks, use the course content to create your own security tester
• Choose domain, techniques as you like
• Submission due after two more weeks
• Will be graded for creativity and efficiency