Software Security Analysis

Chair of IT Security / I20
Prof. Dr. Claudia Eckert
Technical University of Munich

Fabian Franzen
franzen@sec.in.tum.de

Fabian Kilger
fkilger@in.tum.de

Alexander Küchler
alexander.kuechler@aisec.fraunhofer.de

Konrad Weiss
konrad.weiss@aisec.fraunhofer.de

Florian Wendland
florian.wendland@aisec.fraunhofer.de

Maximilian Kaul
maximilian.kaus@aisec.fraunhofer.de

July 13, 2020

What this seminar is about?

- Modern Sofware consists out many software components

What this seminar is about?

- Modern Sofware consists out many software components
- This software components can contain easyily contain about 100.000 lines of code
  - e.g. OpenSSL has about 230000 LOC
  - the linux kernel even has about 1.4 million LOC

What this seminar is about?

- Modern Sofware consists out many software components
- This software components can contain easyily contain about 100.000 lines of code
  - e.g. OpenSSL has about 230000 LOC
  - the linux kernel even has about 1.4 million LOC

- Is this secure?

Examples where it was not...

Apple Goto Fail

```
...
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto ↓fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto ↓fail;
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;

err = sslRawVerify(ctx,
                   ctx->peerPubKey,
                   dataToSign,              /* plaintext */
                   dataToSignLen,           /* plaintext length */
                   signature,
                   signatureLen);
```

Do you remember other *accidents*?

Software Analysis Techniques

An overview of automated software analysis techniques:

- Static code analysis
  - Dataflow analysis
  - Abstract interpretation
  - RegEx search for secret values
- Dynamic code analysis
  - Code Sanitizer (z.B. AddressSanitizer von Clang)
  - Fuzzing
  - Symbolic Execution
  - Binary Instrumentation

String Reconstruction

Spot the error:

```
1  public void printAddresses(int id) throws SQLException {
2      Connection con = DriverManager.getConnection("students.db"
           );
3      String q = "SELECT addr FROM address";
4      if (id!=0)
5          q = q + "WHERE studentid=" + id;
6      ResultSet rs = con.createStatement().executeQuery(q);
7      while(rs.next()){
8          System.out.println(rs.getString("addr"));
9      }
10 }
```

- *String reconstruction* is the problem of creating a set of *all possible values* that may be assigned to a string variable at a program location
- Extremely useful when search for vulnerabilities in programs: SQL injections, unsafe crypto ciphers, reflection, etc.
- Different interesting approaches to the problem: abstract interpretation, set constraints, context-free grammar generation

Discover Security Vulnerabilities through Machine Learning supported Static Analysis

- Software projects are becoming larger in size and in number
- Large scale manual auditing becomes less feasable
- Idea: Mine open source code and known vulnerabilities, use machine learning techniques to train and classify pieces of code as potentially vulnerable.
- Is enabled by decades of vulnerability documentation and easy to access OpenSource code.

Course Organization

We will organize the seminar like a scientific conference. You will present your research in written and in a presentation to your peers.

The paper you will be writing will (most likely) be a *Systematization of Knowledge (SoK)* or *introductory* paper.

SoK papers do not propose a novel approach. They take a broader view on a topic, explain the core concepts and put the most relevant works in context.
Introductory papers explain the core concepts of a field, the problems they are applied to and ongoing research directions.

Course Organization

- Research & Paper Writing
  - Write a scientific paper of (exactly) 10 pages (excluding references and appendices)
  - We will use the standard Usenix Security LaTeX template
- Review Phase
  - Every participant creates 2-3 reviews of her/his peers
  - 1 page
- Rebuttal & "Camera Ready" Phase
  - Integrate the reviewers' remarks, improve your paper as far as possible
  - Submit the "camera ready" version (final polished version)
  - Write a *rebuttal*, i.e. a response summarizing how you addressed the reviewers' remarks
- Presentation
  - 30 minutes presentation
  - 15 minutes discussion
- Language: English

Time Table (Draft!!!)

Today	Topic Presentations. *Register for this seminar until 21.07.2020.*
16.09.2020	Start of topic assignments
16.11.2020	Session: How to write a research paper?
02.12.2020	Submit your first version (outline finished, 80% of content) for review
16.12.2020	Submit Reviews
13.01.2021	Submit your rebuttal + "camera-ready" version + presentation
??.+??.01.2021	**Meeting**: Presentations and discussion

Requirements

"First version" Structure & main contents of the paper are fix. Introduction, conclusion, abstract might not be fully finished. Language does not have to be perfect, graphics might not be finished, some references might be missing. Focus on the "meat" of the paper!

"Review" Provide *constructive* feedback on your fellows' papers. //

"Rebuttal" Your answer to the reviewer. Explain which suggestions you incorporated in the final version. If you do not agree with any suggestions, provide a short justification.

"Camera Ready" The *perfect* and final version of your paper that you and your reviewers will be happy with. Correct formatting, correct citations, no typos.

Grading

The grading is composed of *mandatory* and *graded* parts:

Mandatory:

1. Timely submission of paper, reviews, final paper
2. Participation in discussions

Graded:

1. Paper (50%)
2. Review (10%)
3. Experiments (15%)
4. Presentation (25%)

Location

- To be honest: We do not know yet, because of Covid-19
  - If onsite teaching is possible, in a room at TUM or Fraunhofer AISEC
  - Otherwise: Online via BBB

Registration

- Registration using the matching system
- No letter of motivation
- Instead: Try to analyze a few selected software repositories automatically for bugs.
  - e.g use AFL, Clang Static Analyzer, angr, KLEE
- Details will be provided on course website!
- Provide us with a write-up with security bugs you found mentioning its location and how you found it!
- Mail write up to `franzen@sec.in.tum.de`

Q&A