Finding vulnerabilities

Secure Programming
Lecture 4

Reading assignment

Read (before next Monday)

William Halfond and Alessandro Orso,
AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks,
Automated Software Engineering (ASE) 2005

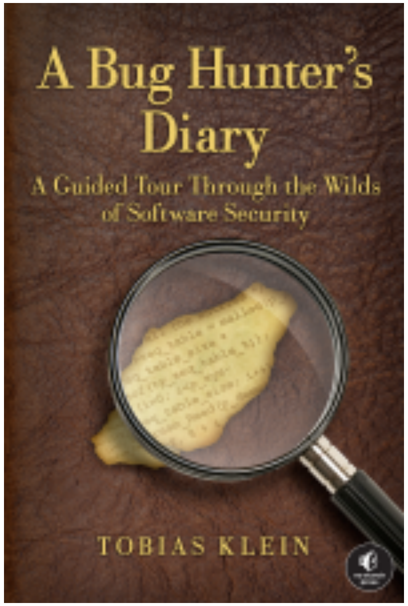
Where are we?

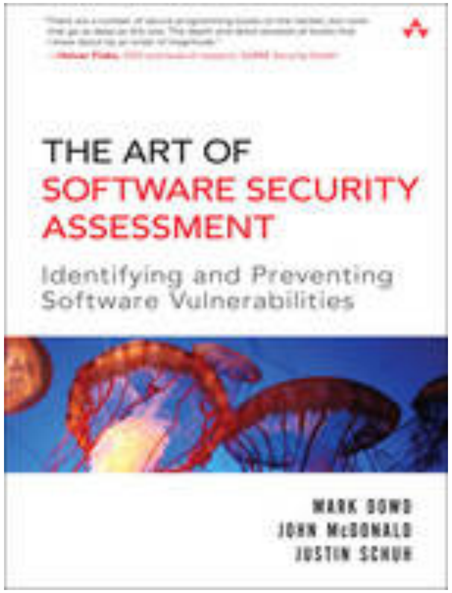
Set of principles that can guide us when designing secure systems, but…

- Not all systems designed with these principles in mind
- Principles are not a silver bullet

How do we go about finding vulnerabilities?

A couple of great references





Security assessment

Comprehensive review of the security of a system:

- Design review
- Operation review
- Application review

Design review

Review of design documents:

- Identify vulnerabilities in architecture
- Prioritize components for implementation review

Threat modeling

Several steps:

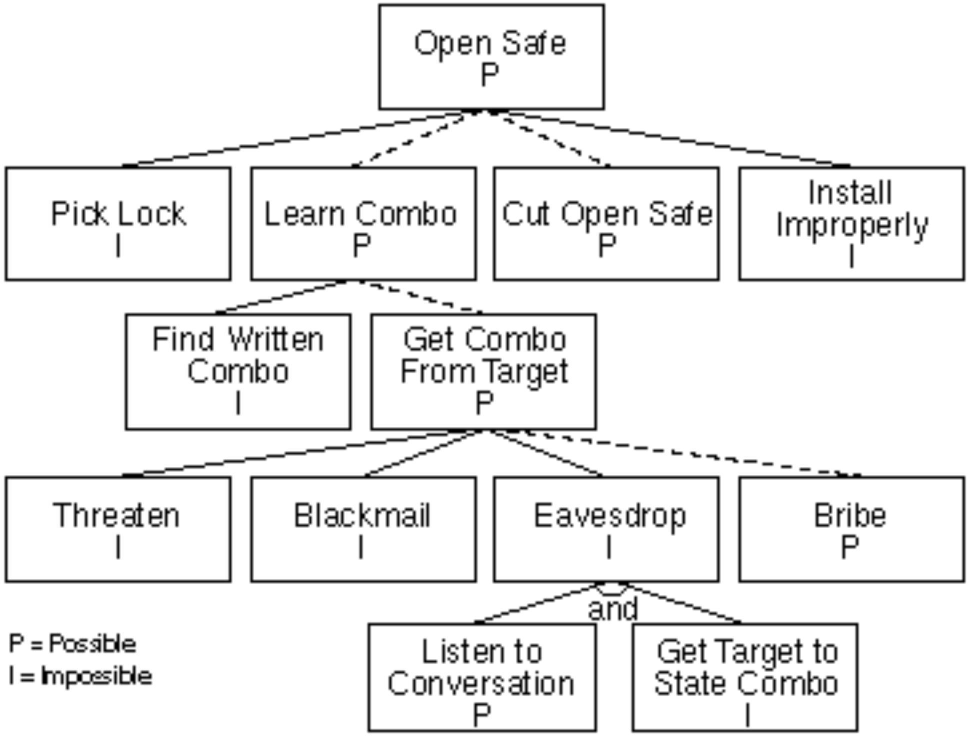
- Information collection (assets, entry points, external entities, trust levels, main components)
- Architecture modeling
- Threat identification (e.g., attack trees)
- Documentation of findings
- Prioritization of implementation review (e.g., DREAD)

Attack trees

- Formal methodology for analyzing the security of systems
- Understand different ways in which system can be attacked
- Understand who the attackers are, and their abilities, motivations, and goals

→ design proper countermeasures

Attack trees



Taken from B. Schneier, [Attack trees](), Dr. Dobb's Journal, 1999

DREAD risk rating

You may find a lot of issues → prioritization is key

- Damage: how bad is the problem?
- Reliability: how reproducible is the attack?
- Exploitability: how much work is needed to launch the attack?
- Affected users: how many?
- Discoverability: how hard is it to spot the vulnerability?

DREAD problems

Original rating obtains overall risk value by averaging the score in each category. Reasonable?

- Risk A: damage 1, rest 10 → 8.2
- Risk B: discoverability 1, rest 10 → 8.2

Which risk would you prefer?

Operational review

Find vulnerabilities in the way a system is configured or deployed:

- Review the *attack surface*
- Identify insecure defaults
- Detect unnecessary services

Related concept: system “hardening”

Application review

Review of the actual software artifact, incorporating:

- Results from the design review
- Results from the operational review

Several techniques:

- Code audit
- Testing techniques (e.g., fuzzing)

Preliminaries

- What are the *review's* goals
- What is the timeline?
- Ensure proper legal documents are in place!

Tips

- Avoid drowning
- Iterative process
- Coordinate with other assessors
- In the end, creative process that requires to acquire and build specialized skills

Code auditing strategies

- Code comprehension strategies:
  - Trace malicious input
  - Analyze sub-components (module, algorithm, class)
- Candidate point strategies:
  - Trace back from potential vulnerabilities (from source code analysis or black box testing tools)
- Design generalization strategies:
  - Model the system: infer high-level abstractions and identify vulnerabilities
  - Hypothesis testing: model subsystem starting from an hypothesis of its working

Fuzzing

Throw random/unexpected/corner case input to an application and see if manifests a bug (e.g., crashes)

- Fast: checks for lots of cases
- (Relatively) simple: enables testing of cases that are not manageable via manual code audit
- Effective

Well-known technique
(B. Miller's original 1990 report)

Fuzzing

More recent efforts:

- Fuzzing frameworks (e.g., Dave Aitel's SPIKE)
- Grammar-based generation of inputs
- Stateful fuzzers
- Whitebox fuzzing (e.g., Microsoft's SAGE)

Case study

Finding vulnerabilities in electronic voting systems

D. Balzarotti, G. Banks, M. Cova, V. Felmetsger, R. Kemmerer, W. Robertson, F. Valeur, G. Vigna,

An Experience in Testing the Security of Real-World Electronic Voting Systems,
IEEE Transactions on Software Engineering, 36(4), 2010

Top-To-Bottom Review (TTBR)

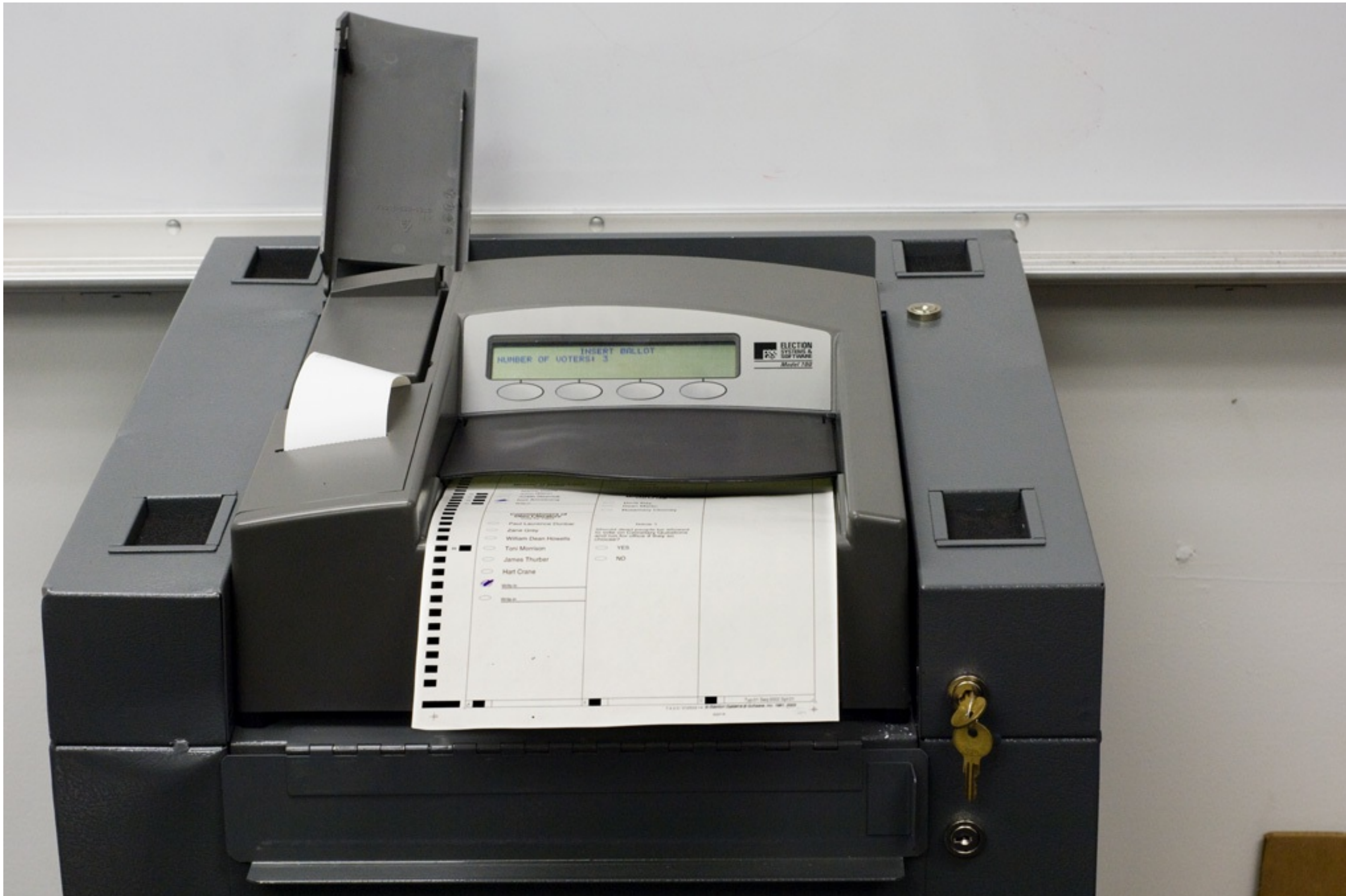
- Review of electronic voting systems ordered by California Secretary of State D. Bowen in summer 2007
- Similar study in Ohio the following year
- “Are our voting systems secure, accurate, reliable and accessible?”
- For each analyzed system, established:
  - Document review team
  - Source code review team
  - “Red” team

Analysis environment

All teams required to sign strict NDAs
→ I cannot show you any piece of code

Voting system: DREs and VVPAT

Voting system: optical scanners



Voting system: DTDs

Functional overview

- Prior to election, ballot information is prepared at election central
- On election day, voting machines are initialized using Data Transport Devices (DTDs)
- DREs and optical scanners are tested with sample votes (logic and accuracy testing)
- Actual voting takes place
- After election is closed, results are collected on DTDs and returned to election central
- Tally is computed

Security evaluation

Scope of work: try and compromise the accuracy, security, and integrity of the voting systems

- Cause incorrect recording, tabulation, tallying or reporting of votes
- Alter critical election data such as election definition or system audit data

We were provided with:

- Documentation
- Source code
- Working machines

Methodology

High-level view of the system:

- Information gathering
- Identify high-level components and information flows
- Develop misuse cases

Low-level, concrete implementation:

- Low-level information flow
- Identify threats and attack exposures
- Attack a component
- Compromise the entire system

0. Information gathering

Collect all available information on the system and set up the testing and analysis environment

- Copy of each components
- Copy of source code and binaries
- Copy of all documentation
- Vendor support (e.g., training)

1. High-level components and flows

- Identify abstract components (as opposed to actual, physical machines): e.g., DRE, DTD
- Identify high-level information that is generated, transported, or used by each component: e.g., ballot, recorded vote
- Security assumptions (confidentiality, integrity, availability) about this information:
  - Data loaded on DTD is the same as that generated at election central
  - Data on DTD cannot be altered

2. Misuse cases

- Devise scenarios where some security assumptions is violated and determine the resulting failure
- What-if scenarios
- Data on DTD can be altered without being detected:

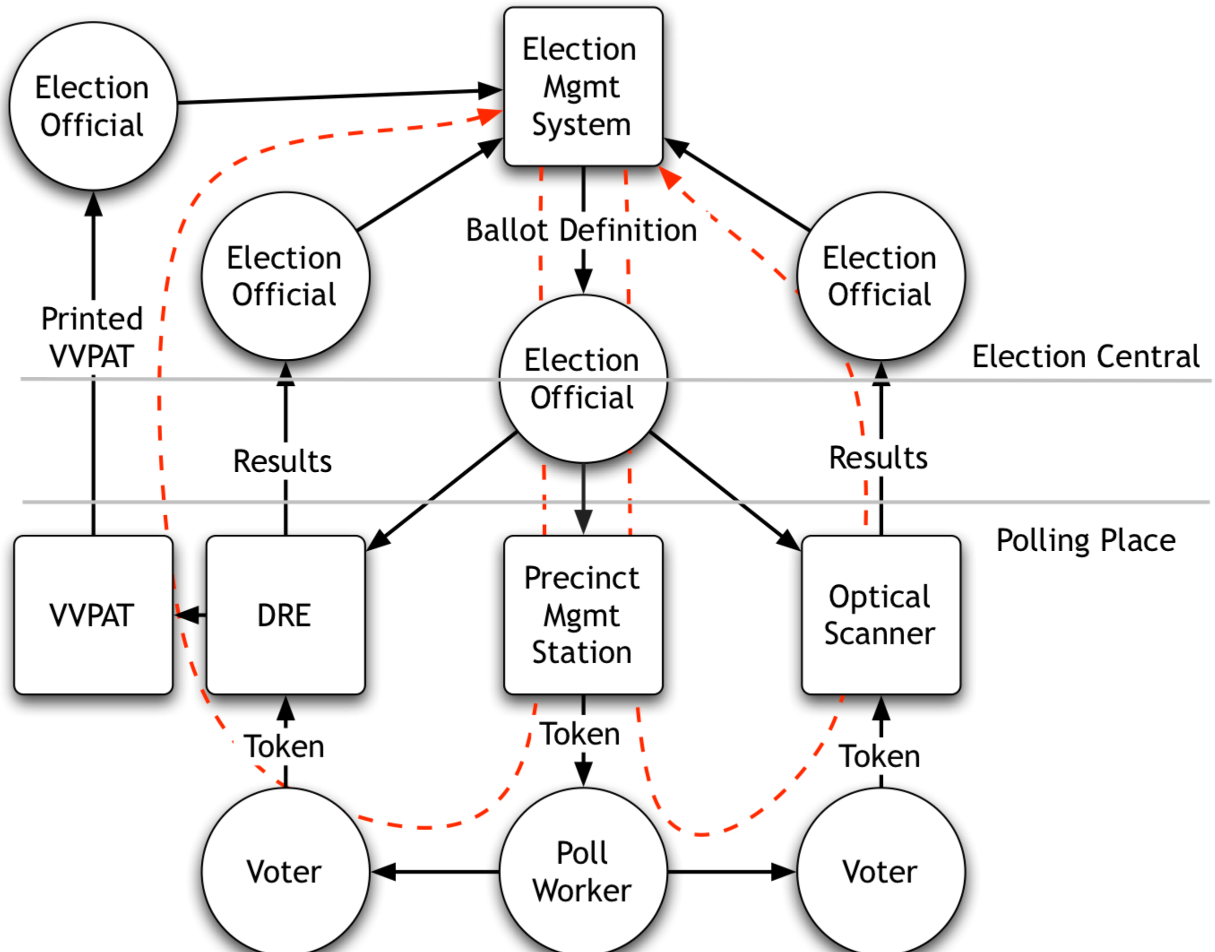
  → invalid ballot information

  → invalid election results

3. Low-level information flow

- Model input/output interface of each software and hardware component
- Identify data, protocol, data format, and physical carrier
- Determine how data is authenticated and validated by each component and how it is protected from eavesdropping, MITM attacks, tampering and replay attacks
- Understand how encryption is used and how key material is managed

Components and information flow



Threats and exposures

Threat modeling

- Who are the attackers?
- What are their motivations, capabilities, and goals?

Identify actual attack scenarios, combining

- Threat modeling,
- Misuse cases (step 2), and
- Low-level information (step 3)

Attacking a component

- Vulnerability analysis of a component
- Exploit development
  - Ad hoc tools (debuggers, exploitation frameworks, etc.
  - Long days (and nights) in the lab :-)

Compromising the system

- Leverage the vulnerabilities identified earlier to compromise the entire system
- Evaluate how a compromised component can take advantage of legitimate information flow to take control of other devices

Findings

- Autorun vulnerability on election management system at election central

  → arbitrary code execution
- Integer overflow vulnerability in DRE

  → arbitrary code execution
- Voter cards are encrypted but key is stored on the card itself

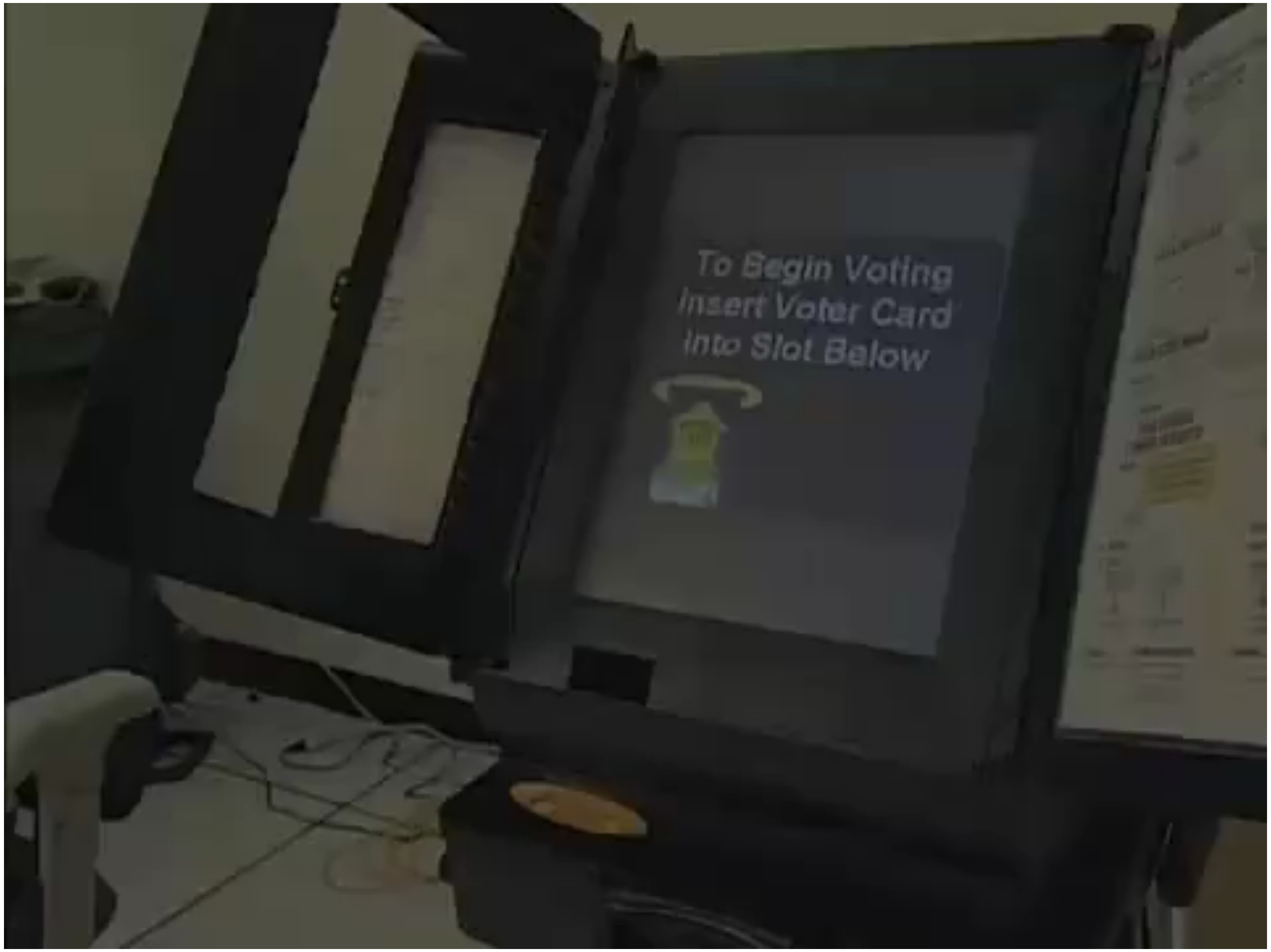
  → multiple voting
- DRE store in a global variable whether it's in testing mode

  → detection evasion
- many, many more!

An election-stealing virus

1. By leveraging the autorun vulnerability, install a Trojan on the election management system
2. The Trojan modifies the DTD data to exploit the integer overflow vulnerability and install a malicious firmware on the DRE
3. The malicious firmware modifies the vote, causes denial of service attacks, and disrupts the elections

One attack



http://www.youtube.com/watch?v=moEsgdzZ19c&t=3m14s

Take away points

Security assessment can look more like an art than science

- There are processes, methodologies, and tips to help
- Whole system analysis: design, operations, code
  - Lots of work
  - Need to know analyzed system in and out
- Focus on assessment's goals, keep timeline in mind