School of Computing
FACULTY OF ENGINEERING & PHYSICAL SCIENCES

UNIVERSITY OF LEEDS

COMP3911 Secure Computing

17: Managing & Avoiding Vulnerabilities

Nick Efford

https://comp3911.info

Objectives

UNIVERSITY OF LEEDS

To explore key issues relating to vulnerabilities:

- How are they discovered, categorised, publicised?
- How do we find them in existing code?
- How do we best avoid introducing them in the first place?

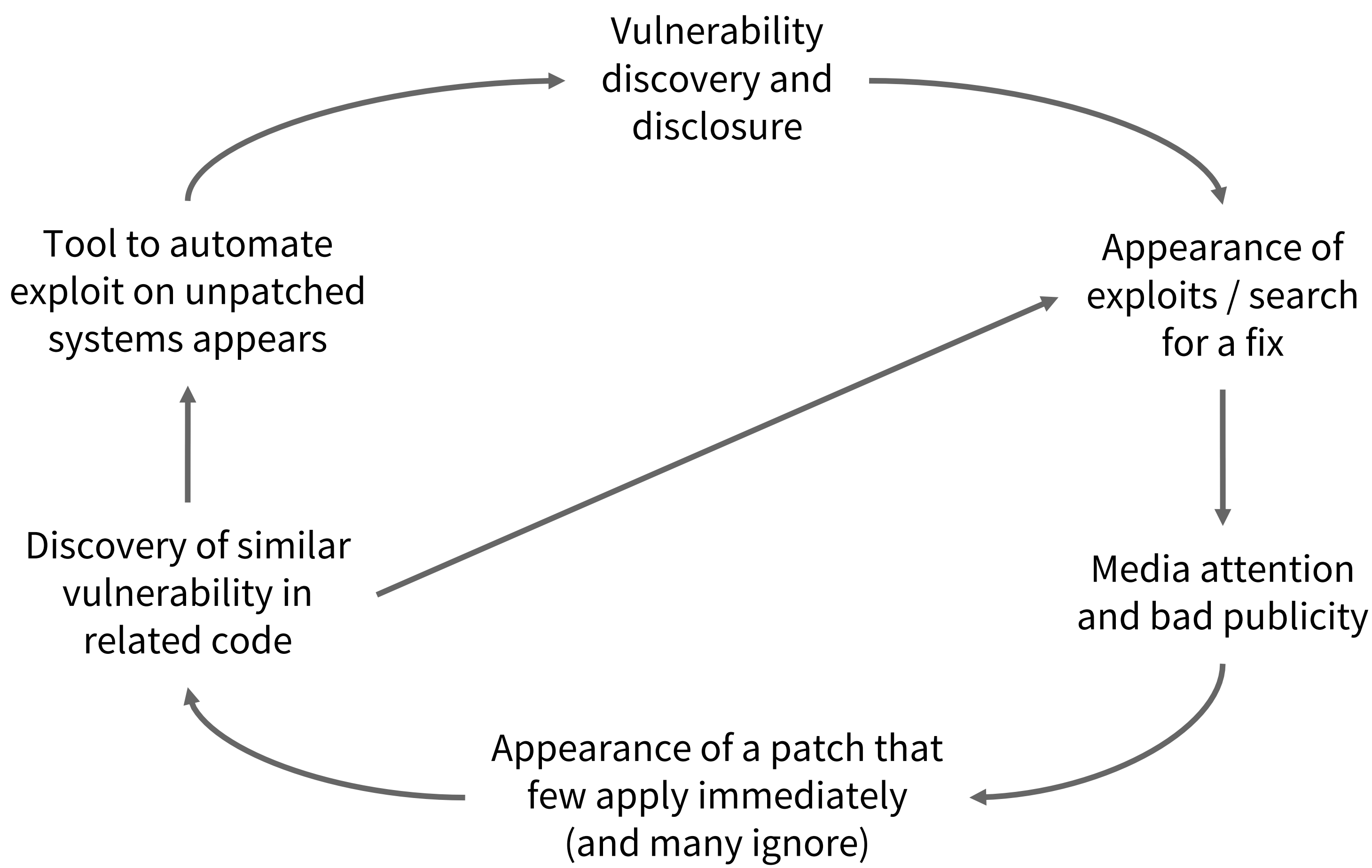
Vulnerabilities

UNIVERSITY OF LEEDS

- *Potential* security weaknesses in a system
- Attacks that use a vulnerability are **exploits**; not all vulnerabilities turn into realistic exploits
- 'Arms race' between security researchers and attackers: who can discover an exploitable vulnerability first?
- Researchers practice **responsible disclosure**
    - Inform software vendor of vulnerability before going public with it after a given time period (Why?)
- **0day** ('zero day') = exploit that is being actively used *before* researchers/vendor discover it

The Vulnerability Cycle

UNIVERSITY OF LEEDS



Standardisation

UNIVERSITY OF LEEDS

ISO 29147 provides guidelines for vendors on receiving vulnerability reports from external researchers and disclosing remediation information

ISO 30111 describes a standard for processes that vendors can follow between receipt of a report and disclosure

Aids to Discussion/Analysis

UNIVERSITY OF LEEDS

- Common Vulnerabilities & Exposures (CVE)
  - **CVE identifier** scheme allows vulnerabilities to be uniquely numbered, simplifying tracking
- Common Weakness Enumeration (CWE)
  - Standard for classifying vulnerabilities *by type*
  - Example: CWE-120 = ‘Buffer copy without checking size of input’ (classic buffer overrun)
- Scoring systems
  - Useful for prioritising vulnerabilities
  - Competing standards: **CVSS** and **CWSS**

Dissemination

UNIVERSITY OF LEEDS

- US-CERT
  https://us-cert.cisa.gov
- UK’s National Cyber Security Centre
  https://www.ncsc.gov.uk
- BugTraq mailing list
  https://www.securityfocus.com/archive/1
- Software vendors
  (ideally with accompanying patches!)
- etc…

Questions

- How do we find vulnerabilities in existing code?
- How do we minimise the risk of them being introduced into our software in the first place?

Source Code QA Techniques

UNIVERSITY OF LEEDS

- Code review
- Checklists
- Static analysis tools

Code Review

UNIVERSITY OF LEEDS

- Exploits the ‘Hawthorne Effect’
- Various approaches
  - Pair programming (in XP)
  - Peer review
  - External review
- Independent reviewers are more objective
- Reviews can be tied to commits/pushes
  - VC tools can help – e.g., by highlighting diffs

The ‘Many Eyeballs’ Fallacy

UNIVERSITY OF LEEDS

Linus’ Law:

“Given enough eyeballs, all bugs are shallow”

(Eric Raymond, The Cathedral And The Bazaar, 1999)

- Lack of convincing evidence for this
- Research shows that bug finding rates in OSS do *not* scale linearly with number of reviewers
- Caveat: you need the ‘right kind of eyeballs’ to uncover security-related bugs!

Today In Infosec
@todayininfosec

1989: Brian Fox introduced code into Bash, later released as version 1.03, which included the first of the Shellshock vulnerabilities publicly reported 9,169 days later. That's 25 years, 1 month, and 13 days of exploitability.

Takeaway? You're always running exploitable code.

Checklists

UNIVERSITY OF LEEDS

- Help to ensure proper consideration of security issues during programming or a code review
- Also useful during design and security testing
- Can be signed off formally
- Cannot address every security issue
- Need to be maintained to remain useful
    - Put them under version control!

Static Analysis Tools

UNIVERSITY OF LEEDS

- Lexical analysis
  - Flawfinder (C, C++)
  - RATS (C, Perl, PHP, Python, Ruby, OpenSSL)
- Parsing
  - Splint – 'Secure Programming Lint' (C only)
- Data flow analysis
  - FindSecBugs – plug-in for SpotBugs (Java only)
- 'Separation logic and bi-abduction'
  - Facebook's Infer (C, Objective-C, Java)

Example


```c
#include <stdio.h>
#include <string.h>

int main(int argc, char* argv[])
{
  char buf[256];

  if (argc > 1) {
    strcpy(buf, argv[1]);
    printf(buf);
    printf("\n");
  }

  return 0;
}
```

What might a static analysis tool warn you about here?

More Examples

UNIVERSITY OF LEEDS

```
if (access(file, W_OK) == 0) {
  ...
  fp = fopen(file, "wb+");
  writeToFile(fp);
}
```

'Time Of Check, Time Of Use' vulnerability

```
#include <iostream>
#include <cstdlib>

int main()
{
  for (int i = 0; i < 10; ++i) {
    std::cout << random() << '\n';
  }
  return 0;
}
```

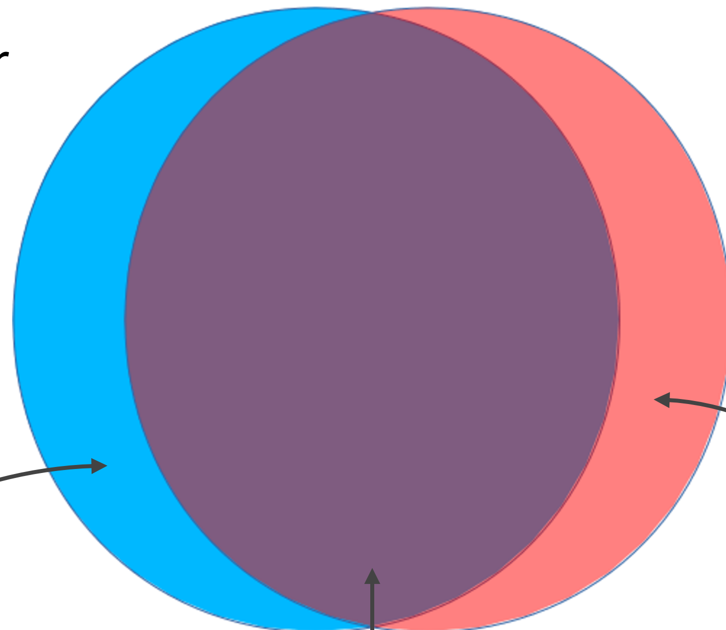
Limitations of Static Analysis

UNIVERSITY OF LEEDS

- False positives reported
    - Danger of ‘flaw fatigue’
- False negatives
- Flaw database must be maintained
- Source code required

Testing != Security Testing

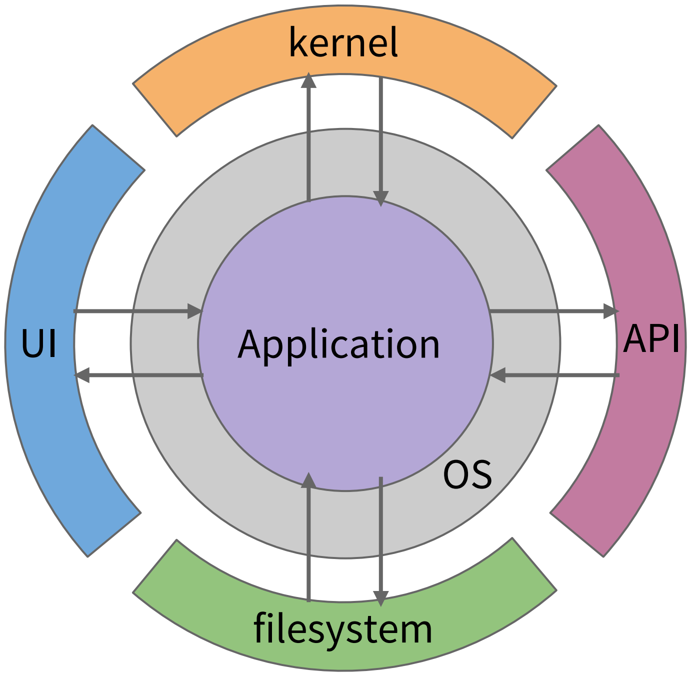
UNIVERSITY OF LEEDS



Whittaker’s Fault Model

UNIVERSITY OF LEEDS

Developers and testers tend to focus on user interface input…



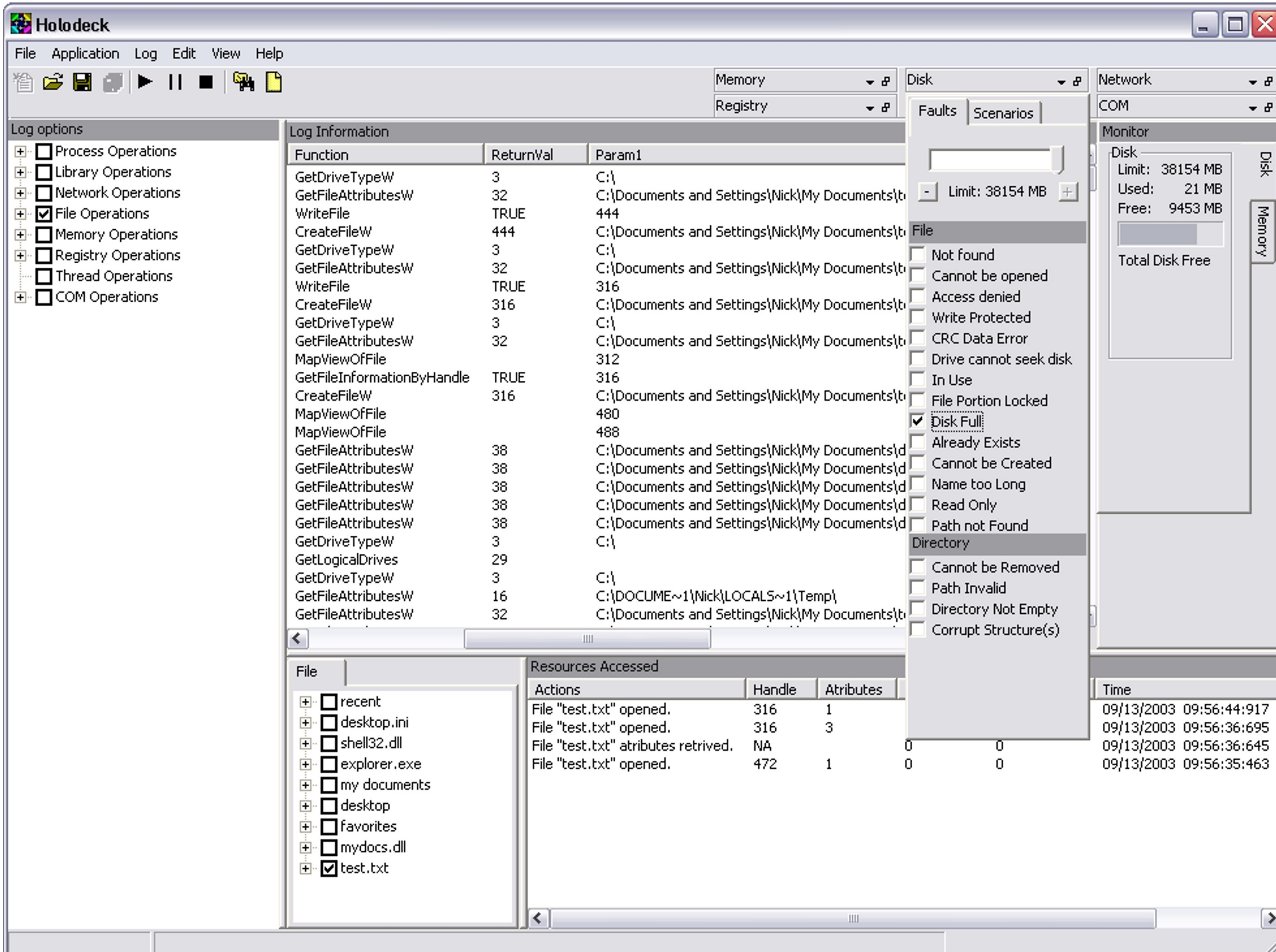
… but other less visible components also provide input to an application

Security testing must investigate effects of faults in *all* input sources

Run-time Fault Injection

UNIVERSITY OF LEEDS

- Can simulate faults using software that intercepts and modifies system calls
- Example: **Holodeck** (Windows systems)
  - System monitoring
  - Fault injection
    - Insufficient memory, failure to lock memory
    - No disk space, too many open files, no disk in drive
    - Low bandwidth, network disconnected, no ports
    - Missing libraries…


Holodeck
File Application Log Edit View Help
Memory
Registry
Disk
Faults Scenarios
Network
COM
Log options
Process Operations
Library Operations
Network Operations
File Operations
Memory Operations
Registry Operations
Thread Operations
COM Operations
Log Information
Function ReturnVal Param1
GetDriveTypeW 3 C:\
GetFileAttributesW 32 C:\Documents and Settings\Nick\My Documents\t
WriteFile TRUE 444
CreateFileW 444 C:\Documents and Settings\Nick\My Documents\t
GetDriveTypeW 3 C:\
GetFileAttributesW 32 C:\Documents and Settings\Nick\My Documents\t
WriteFile TRUE 316
CreateFileW 316 C:\Documents and Settings\Nick\My Documents\t
GetDriveTypeW 3 C:\
GetFileAttributesW 32 C:\Documents and Settings\Nick\My Documents\t
MapViewOfFile 312
GetFileInformationByHandle TRUE 316
CreateFileW 316 C:\Documents and Settings\Nick\My Documents\t
MapViewOfFile 480
MapViewOfFile 488
GetFileAttributesW 38 C:\Documents and Settings\Nick\My Documents\d
GetFileAttributesW 38 C:\Documents and Settings\Nick\My Documents\d
GetFileAttributesW 38 C:\Documents and Settings\Nick\My Documents\d
GetFileAttributesW 38 C:\Documents and Settings\Nick\My Documents\d
GetFileAttributesW 38 C:\Documents and Settings\Nick\My Documents\d
GetDriveTypeW 3 C:\
GetLogicalDrives 29
GetDriveTypeW 3 C:\
GetFileAttributesW 16 C:\DOCUME~1\Nick\LOCALS~1\Temp\
GetFileAttributesW 32 C:\Documents and Settings\Nick\My Documents\t
Limit: 38154 MB
File
Not found
Cannot be opened
Access denied
Write Protected
CRC Data Error
Drive cannot seek disk
In Use
File Portion Locked
Disk Full
Already Exists
Cannot be Created
Name too Long
Read Only
Path not Found
Directory
Cannot be Removed
Path Invalid
Directory Not Empty
Corrupt Structure(s)
Monitor
Disk
Limit: 38154 MB
Used: 21 MB
Free: 9453 MB
Total Disk Free
Disk
Memory
File
recent
desktop.ini
shell32.dll
explorer.exe
my documents
desktop
favorites
mydocs.dll
test.txt
Resources Accessed
Actions Handle Atributes Time
File "test.txt" opened. 316 1 09/13/2003 09:56:44:917
File "test.txt" opened. 316 3 09/13/2003 09:56:36:695
File "test.txt" atributes retrived. NA 0 0 09/13/2003 09:56:36:645
File "test.txt" opened. 472 1 0 0 09/13/2003 09:56:35:463

Testing Using Whittaker's Model

UNIVERSITY OF LEEDS

- Attack environmental dependencies
  - Block access to libs; manipulate Windows registry; corrupt config files; simulate lack of memory, disk, etc
- Attack user interfaces
  - Overrun buffers; use escape characters; inject code
- Attack the design
  - Find unprotected accounts; connect to all ports; fake sources of data; explore alternate routes to functionality
- Attack the implementation
  - Exploit TOC-TOU; force all errors; uncover test APIs; screen temporary files for sensitive information

Using Threat Models

UNIVERSITY OF LEEDS

- Threat model drives the testing process
- Each threat needs a test plan
- STRIDE tells us what types of test to perform
- Quantitative risk assessment (e.g., DREAD rating) can be used to prioritise test plans

Secure Software Engineering?

1. Analyse requirements
2. Produce a design
3. Implement the design
4. Add security
5. Test
6. Fix bugs
7. Ship product

What's wrong with this plan?
(aside from being classic Waterfall...)

‘Secure From Day One’

UNIVERSITY OF LEEDS

- Security (re)training for team members
- Threat model begun at the earliest stages, when product is still being envisioned, and refined thereafter
- Design process is driven by threat model and adopts standard principles such as reluctance to trust, granting of minimal privilege, etc
- Designs undergo security reviews
- Code QA and testing as described earlier
- ‘Security push’ in later stages
- Full security audit before shipping

Summary

UNIVERSITY OF LEEDS

We have

- Discussed the vulnerability cycle, and tools that help in the discussion and analysis of vulnerabilities
- Explored the roles played by code review and testing in the prevention of vulnerabilities
- Considered the value of static analysis tools
- Discussed how software engineering processes need to change in order to address security concerns

Follow-Up / Further Reading

UNIVERSITY OF LEEDS

- CVE identifiers & CVE Numbering Authorities
- Common Weakness Enumeration & CWSS
- ZDNet article: "Google Project Zero: 95.8% of all bug reports are fixed before deadline expires"
- Whittaker JA & Thompson HH, *How to Break Software Security*, Pearson 2004
- Holodeck source code on GitHub
- libfiu – C library for fault injection in POSIX API