Term Project

All submissions should follow the 2-column format for IEEE conference papers.

• Proposal and sources: Oct 10

• Semi-final report: Nov 7
  – It should indicate that you have finished at least two-thirds of the work. It should include an abstract, discussion of background literature, a summary of the investigations/findings, any refinements of the proposal objectives as a result of the past study, what the final report will contain and the applicable references.
  – Technical details, equations/tables/plots/screen-shots
  – You must be aware of the current trends in research/industry.

• Slides: Due Nov 18 (more details later)

• Ten-minute oral presentation Nov 19-Dec 8
Peer Interaction

• Your interaction with other student’s research is a part of the class.
• You will need to review semi-final report of two fellow students
  – Identify main contributions
  – Strength/weaknesses
  – Suggestions for improvements
  – Suggested additional references
• Presentations: reviews and comments
• Your interaction will be evaluated
Quantitative Cyber-Security
Colorado State University
Yashwant K Malaiya
CS559
Projects

CSU Cybersecurity Center
Computer Science Dept
Research

• Understand the techniques and results in a chosen field
  – Examine articles from diverse sources
  – Study selectively

• Identify current status, trends, unexplored issues

• Search for information
  – Multiple types of sources
  – Multiple key words

• Search “around” an article
  – Backward search: citations
  – Forward search: cited by (Google scholars)
  – Horizontal search: related publications
Search Databases

Specific sources: database indexes

- Google Scholar
  - Forward links: [Paper X Cited by](#)
  - Backward Links: [Paper X cites](#)

- Researcher sites
  - Personal/Group Website
  - DBLP
  - Google Scholar: [researcher](#)

- CSU Library etc.

General (accessible through CSU Library)
- ACM Digital Library
- IEEEXplore Digital Library
- ScienceDirect etc
Source types

• News (such as Google News)
• Conferences: held once a year, proceedings published
  – Conference, Symposium, ...
• Industry publications
  – Magazines, blogs, white papers, product website
• Journals: published several times a year
  – Rigorously reviewed, long publication delay
  – Journal, Transactions, ...
• Research groups
  – Industry, academic, consultants: web site
• Books: often well-known stuff
  – Research updates: monographs
How to Read Papers: THE THREE-PASS APPROACH

• The first pass: Read (a large number)
  – the title, abstract, and introduction
  – section and sub-section headings, but ignore everything else
  – the conclusions

• The second pass: Read (an intermediate number)
  – figures, diagrams and other illustrations
  – mark relevant unread references for further reading
  – Do you need to read it in detail?

• The third pass: Read critically (closely related)
  – identify and challenge assumption and views
  – Loop up references needed

Evaluation of Research

Similar to paper review for conferences/journals

• Significance and originality
  – Timeliness
  – Originality

• Thoroughness of research
  – Familiar with the field?
  – Has seen significant/recent papers?

• Depth of understanding displayed

• Presentation
You Must Do Research

Not enough:
• Summary of a couple of papers
• Summary of work of a single research group
• Rephrasing of existing surveys

You must know (and should be able to answer related questions):
• Current state of the art
• Alternative approaches and how they can be evaluated
• Technology trend
• Find data describing the technology
• Existing issues and challenges
Citing Sources

“IEEE” “ACM” etc:

• These are professional organizations that organize numerous conferences and published journals
• You must specify the author, title of paper, specific names of conference/journal, associated details, date, page numbers
• A simple URL is not a valid citation
• URL not needed for conference, journal publications. Needed for on-line publications (Organizational reports, Industrial white-papers, News etc)


You must include

• Title, your name, class, year, professor’s name
• Abstract: What does it include and why is it important
• Background: Other existing work and background ideas
• Technical discussion: detailed presentation of findings with non-text material (equations, charts, plots, tables, algorithms etc.)
• Discussion of results
• Summary, future work
• References
• Appendix if any
<table>
<thead>
<tr>
<th>Name</th>
<th>Project Number</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dubois, Alexandre</td>
<td>1</td>
<td>economic tradeoffs due to security issues</td>
</tr>
<tr>
<td>Li, Jacinda</td>
<td>6</td>
<td>Analysis of Electronic Payment Systems</td>
</tr>
<tr>
<td>Pineiro Rivera, Luis</td>
<td>6</td>
<td>Security of Payment Systems</td>
</tr>
<tr>
<td>Mulligan, Brett</td>
<td>10</td>
<td>Fuzzing Open Source IoTProject</td>
</tr>
<tr>
<td>Chen, Sirius</td>
<td>11</td>
<td>secure containers</td>
</tr>
<tr>
<td>Liu, Zijuan</td>
<td>11</td>
<td>Security in Virtualized Systems</td>
</tr>
<tr>
<td>Al Amin, Md</td>
<td>12</td>
<td>Ransomware</td>
</tr>
<tr>
<td>Neumann, Don</td>
<td>12</td>
<td>Ransomware</td>
</tr>
<tr>
<td>Haynes, Katherine</td>
<td>13</td>
<td>DeepNeural Networks to Improve Phishing Detection</td>
</tr>
<tr>
<td>Rodriguez, Luis</td>
<td>13</td>
<td>Quantitative Examination of Phishing</td>
</tr>
<tr>
<td>Zhao, Qingyi</td>
<td>13</td>
<td>phishing</td>
</tr>
<tr>
<td>Weaver, Austen</td>
<td>14</td>
<td>Cost and Cause of U.S. Government Security Breaches</td>
</tr>
<tr>
<td>Eswaran, Suraj</td>
<td>15</td>
<td>CYBER INSURANCE</td>
</tr>
<tr>
<td>Alqurashi, Saja</td>
<td>19</td>
<td>CS networkusing Mitre ATT&amp;CK</td>
</tr>
<tr>
<td>Gowdanakatte, Shwetha</td>
<td>19</td>
<td>ATT&amp;CKFramework and Vulnerability detection forIndustrial Control System</td>
</tr>
<tr>
<td>Kotian, Siddhi</td>
<td>20</td>
<td>Effectiveness of Penetration Testing</td>
</tr>
<tr>
<td>Padalia, Dhruv</td>
<td>20</td>
<td>effectiveness of Penetration Testing</td>
</tr>
<tr>
<td>Shang, Tony</td>
<td>24</td>
<td>deep neural networks to detect DDOS attack</td>
</tr>
<tr>
<td>Cheng, YaHsin</td>
<td>25</td>
<td>Cyber Criminals</td>
</tr>
<tr>
<td>Houlton, Sarah</td>
<td>25</td>
<td>Cyber Crime and Criminals</td>
</tr>
<tr>
<td>Jepsen, Waylon</td>
<td>25</td>
<td>North Korea’s Cyber Criminals</td>
</tr>
<tr>
<td>Petkar, Jayesh Umesh</td>
<td>26</td>
<td>Smartphone Security Model and Vulnerabilities</td>
</tr>
<tr>
<td>Ravichandran, Shree Harini</td>
<td>26</td>
<td>Smartphone Security Model</td>
</tr>
<tr>
<td>Paudel, Upakar</td>
<td>sp</td>
<td>Security Posture of Various Android based IoT</td>
</tr>
</tbody>
</table>
Presentations

• 10-minute presentations, 2 minutes for Q/A/C
  – Suggest max 15 slides
  – Mention your name when asking questions

• One joint project with 2 students
  – 2x presentation time

• Multiple independent projects on the same topic
  – Coordinate to minimize overlap in the presentation
Problem: Password guessing

How easy it is to guess a password?

• If your keyboard has R = 95 unique characters, randomly constructing a password from that whole set, 12-character password, then L = 12.

• $95^{12} = 540,360,087,662,636,962,890,625$ passwords

Entropy $= \log_2(R^L) = 78.9$ bits assuming passwords are created randomly

• Non-randomness makes password guessing easier.
• Measures of password strength proposed and used

Password guessing at login? Can be defeated by

– Limited number of tries: 3-5
– Blocking attempts from unknown/suspected IP addresses
Attacking Salted Passwords

```plaintext
hash()

List of possible password hashes

hash('a8' + word)
List of possible password hashes w/ salt a8

No matches
hashed_and_salted_password.txt

hash('0X' + word)
List of possible password hashes w/ salt 0X

cbw  a8
sandi 0X
amislove hz
bob  K@

cbw  XXXX
sandi YYYY
```
Quantitative Security
Colorado State University
Yashwant K Malaiya
CS 559
Fuzzing

CSU Cybersecurity Center
Computer Science Dept
Fuzzing vs. Testing

User Testing

Run program on many normal inputs, look for bad things to happen

**Goal:** Prevent normal users from encountering errors

Fuzzing

Run program on many abnormal inputs, look for bad things to happen

**Goal:** Prevent attackers from encountering exploitable errors

Ack: Stanford, Columbia
Types of Fuzzing

• **Mutation-based (Dumb) fuzzing**
  – Add anomalies to existing good inputs (e.g., test suite)

• **Generative (Smart) fuzzing**
  – Generate inputs from specification of format, protocol, etc

• **Evolutionary (Responsive) fuzzing**
  – Leverage program instrumentation, code analysis
  – Use response of program to build input set
Mutation-Based Fuzzing

Basic Idea

- Take known good input and add anomalies
- Anomalies may be completely random or follow some heuristics
  - Large integers or strings
  - Randomly flip bits
HTTP Fuzzing Example

Standard HTTP GET Request
GET /index.html HTTP/1.1

Anomalous Requests
GEEEE...EET /index.html HTTP/1.1
GET ///////////index.html HTTP/1.1
GET %n%n%n%n%n%n.html HTTP/1.1
GET /AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA.html HTTP/1.1
GET /index.html HTTTTTTTTTTTTTTTP/1.1
df%w3rasd8#r78jskdflasdjf
4isg8swksdfsksdflsdgmsf$gkjs
Fuzzing PDF Reader

- Download 100s of random PDF files
- Mutate content in the PDF file:
  - flip bits
  - increase size of integers or strings
  - remove data
- Limited by the functionality that the existing files happened to use — unlikely to hit less commonly tested code paths
Mutation-Based Fuzzing

Basic Idea
• Take known good input and add anomalies
• Anomalies may be completely random or follow some heuristics

Advantages
• Little or no knowledge of the structure of the inputs is assumed
• Requires little to no set up time

Disadvantages
• Dependent on the inputs being modified
• May fail for protocols with checksums, challenge-response, etc.
Generation Based Fuzzing

Basic Idea

- Test cases are generated from protocol description: RFC, spec, etc.
- Anomalies are added to each possible spot in the inputs
- Knowledge of protocol should give better results than random fuzzing

```xml
<-- A. Local file header -->
<Block name="LocalFileHeader">
  <String name="lfh_Signature" valueType="hex" value="504b0304" token="true" mut=
  <Number name="lfh_Ver" size="16" endian="little" signed="false"/>
  ...
  <Number name="lfh_CompSize" size="32" endian="little" signed="false"/>
  <Relation type="size" of="lfh_CompData"/>
</Number>
  <Number name="lfh_Decompsize" size="32" endian="little" signed="false"/>
  <Number name="lfh_FileNameLen" size="16" endian="little" signed="false"/>
  <Relation type="size" of="lfh_FileName"/>
</Number>
  ...
  <Number name="lfh_EXTRAFLDLEN" size="16" endian="little" signed="false"/>
  <Relation type="size" of="lfh_EXTRAFLD"/>
  </Number>
  </Block>
</!-- B. File data -->
```
Generation Example: TLS Heartbeat

**Heartbeat** Extension for the Transport Layer Security: to test and keep alive secure communication links without the need to renegotiate the connection each time.
Example: TLS Heartbeat

Heartbleed Vulnerability: server trusts user provided length field and echoes back memory contents following request data
Mutation-based vs. Generation-based

• Mutation-based fuzzer
  – Pros: Easy to set up and automate, little to no knowledge of input format required
  – Cons: Limited by initial corpus, may fall for protocols with checksums and other hard checks

• Generation-based fuzzers
  – Pros: Completeness, can deal with complex dependencies (e.g., checksum)
  – Cons: writing generators is hard, performance depends on the quality of the spec
How much fuzzing is enough?

• Mutation-based fuzzers may generate an infinite number of test cases. When has the fuzzer run long enough?

• Generation-based fuzzers may generate a finite number of test cases. What happens when they’re all run and no bugs are found?

• Sometimes every anomalous test case triggers the same (boring) bug?
In 2010, Charlie Miller fuzzed
  – Adobe Acrobat,
  – Apple Preview,
  – Powerpoint, and
  – Open Office
by downloading PDF and PPT files and five lines of simple fuzzing:

```python
numwrites = random.randrange(math.ceil((float(len(buf)) / FuzzFactor))) + 1
for j in range(numwrites):
    rbyte = random.randrange(256)
    rn = random.randrange(len(buf))
    buf[rn] = '%c' % (rbyte)
```
Charlie Miller’s 5 Lines

Collect a large number of pdf files
– Aim to exercise all features of pdf readers
– Found 80,000 PDFs on Internet

Reduce to smaller set with apparently equivalent code coverage
– Used Adobe Reader + Valgrind in Linux to measure code coverage
– Reduced to 1,515 files of ‘equivalent’ code coverage (Test compaction)
– Same effect as fuzzing all 80k in 2% of the time

Randomly changed selected bytes to random values in files
• Produce ~3 million test cases from 1,500 files

Use standard common tools to determine if crash represents a exploit
• Acrobat: 100 unique crashes, 4 actual exploits 4%
• Preview: 250 unique crashes, 60 exploits (tools may over-estimate) 24%
What if we tried to build tests that try to reach code in the program?

Code coverage is a metric which can be used to determine how much code has been executed.

- **Function coverage**: Has each function in the program been called?
- **Edge coverage**: Has every edge in the Control flow graph been executed?
- **Branch coverage**: Has each branch of each control structure been executed?
- **Predicate coverage**: Has each boolean expression been evaluated to true and false?
Coverage-guided gray-box fuzzing

- Special type of mutation-based fuzzing
  - Run mutated inputs on instrumented program and measure code coverage
  - Search for mutants that result in coverage increase
  - Often use genetic algorithms, i.e., try random mutations on test corpus and only add mutants to the corpus if coverage increases
  - Examples: AFL, libfuzzer
American Fuzzy Lop (AFL)

Seed inputs → Input queue → Next input → Mutation

branch/edge coverage increased?

→ Execute against instrumented target

Add mutant to the queue

Periodically culls the queue without affecting total coverage
Evolutionary Fuzzing

Basic Idea:

Generate inputs based on the structure and response of the program

- **Autodafe**: Prioritizes based on inputs that reach dangerous API functions
- **EFS (Evolving Fuzzer System)**: Generates test cases based on code coverage metrics

Typically instrument program with additional instructions to track what code has been reached — or, if no source is available, track with Valgrind.
Tools

Two influential tools

cross_fuzz — specifically targeted at browser and generating complex DOM sequences

American Fuzzy Lop (AFL) — most everything else
AFL Algorithm

American fuzzy lop (AFL) 2013 initial /2019 stable Michał Zalewski, Google/Snap

- Load user-supplied initial test cases into the queue,
- Take next input file from the queue,
- Attempt to trim the test case to the smallest size that doesn't alter the measured behavior of the program,
- Repeatedly mutate the file using a balanced and well-researched variety of traditional fuzzing strategies,
  - If any of the generated mutations resulted in a new state transition recorded by the instrumentation,
    - add mutated output as a new entry in the queue.
- Go to 2.
Fuzzing challenges

• How to seed a fuzzer?
  – Seed inputs must cover different branches
  – Remove duplicate seeds covering the same branches
  – Small seeds are better.

• Some branches might be very hard to get past as the # of inputs satisfying the conditions are very small
  – Manually/automatically transform/remove those branches
Fuzzing rules of thumb

- Input-format knowledge is very helpful
- Generational tends to beat random, better specs make better fuzzers
- Each implementation will vary, different fuzzers find different bugs
  - More fuzzing with is better
- The longer you run, the more bugs you may find
  - But it reaches a plateau and saturates after a while
- Best results come from guiding the process
- Notice where you are getting stuck, use profiling (gcov, lcov)!