Week 14: Lecture A

Configuration-aware Fuzzing & Course Wrap-up

Monday, April 14, 2025

How are semester projects going?

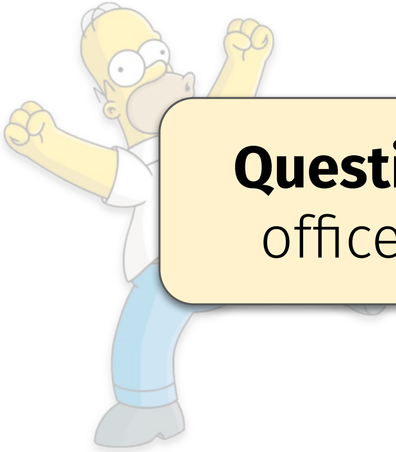
Making progress?

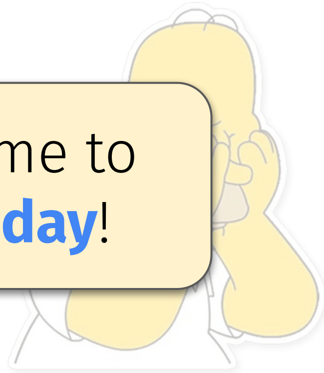
Stuck?

How are semester projects going?

Making progress?

Stuck?





Questions? Need help? Come to office hours **after class today**!

The Next Few Weeks

Part 3: New Frontiers in Fuzzing

Monday Meeting	Wednesday Meeting
Mar. 31 **Kernel Fuzzing** ▶ Readings:	Apr. 02 **LLM-assisted Fuzzing** ▶ Readings:
Apr. 07 **Compiler Fuzzing** ▶ Readings:	Apr. 09 **Hardware Fuzzing** ▶ Readings:
Apr. 14 **Fuzzing Configurable Software** ▶ Readings:	Apr. 16 ⚠ **Final Presentations (Day 1)**
Apr. 21 ⚠ **Final Presentations (Day 2)** ⚠ **Final Reports due Tuesday by 11:59pm via Canvas**	Apr. 23 **No Class (Reading Day)**

Recap: Project Schedule

- **Apr. 16th & 21st:** final presentations
  - **5–8 minute** slide deck and discussion
  - What you did, and why, and what results
  - **Report any bugs found** (and show you did so!)
- What's most important:
  - High-level technique
  - Challenges and workarounds
  - Key results (bugs found, other successes, etc.)
- **Project report** due by midnight last day of class
  - 3–5 pages describing your work and results
  - Reports of any bugs found

Recap: Project Schedule

Final Project (collected via Canvas)

Instructions: Using your skills from Labs 1–3, team-up in groups of two to four students to hunt bugs in a real-world software of your choice! Upon choosing a target, your team must figure out how to (1) harness it, (2) fuzz it, and (3) triage any discovered bugs. You may select any target you like (e.g., popular APIs, video games, kernels), **provided it has *not* been fuzzed before—or has demonstrably not been fuzzed *effectively***. Failure to do your due dilligence will cause your team to lose points—or worse yet—have to find a different target!

Halfway through the semester, your team will present a **5-minute project lightning proposal** to the class outlining your chosen target, your proposed approach, and the significance of your work. At the semester's end, you will prepare and deliver a **final in-class presentation** along with submitting a **final project report** outlining your ultimate approach, findings, and any discovered bugs.

Heilmeier's Catechism will serve as the high-level rubric for your proposal, presentation, and report—so be ready to answer *why* your project idea matters! But most importantly, **get creative and have fun**. Besides Heilmeier's Catechism, other important criterion include:

- **Responsible Disclosure:** Discovered bugs *must* be disclosed to developers! Include any bug report links (e.g., GitHub issues) in your final report. If bug reports are not public, you may include screenshots of your correspondence (e.g., emails) with developers.

Final Reports (3–5 pages) due by **11:59PM** on **Tuesday, April 21st**

Merge your Final Reports and Presentation Slides **into one PDF**

Finalized Team Presentation Schedule

Wednesday

Project Title	**Alter Domus**
Team Members	Braden Campbell, Chandler Turner
Project Title	**Fuzzing Draw.io**
Team Members	Davit Zatikyan, Wilker Gonzalez
Project Title	**Juceing**
Team Members	Pablo Arancibia-Bazan, Alec Carton, Sean McGuirk
Project Title	**Fuzzing PDFio**
Team Members	Adwait Shinganwade, Brensen Villegas
Project Title	**Fuzzing Blender**
Team Members	Austin Garcia, Hyunwoo Lee
Project Title	**Fuzzing MelonDS**
Team Members	Jie Lin, Alexa Fresz, Corinna Healey, Leo Ramirez
Project Title	**Fuzzing Polybar**
Team Members	Kalyan Shankar Ragam, Praneeth Chavva, Koumudi Raju Kothapally, Aparna Gudivada
Project Title	**Fuzzing Vulkan**
Team Members	Ethan Collier, Henry Zheng
Project Title	**Fuzzing Thinger.io**
Team Members	Gavin Dibble, James Hart

Monday

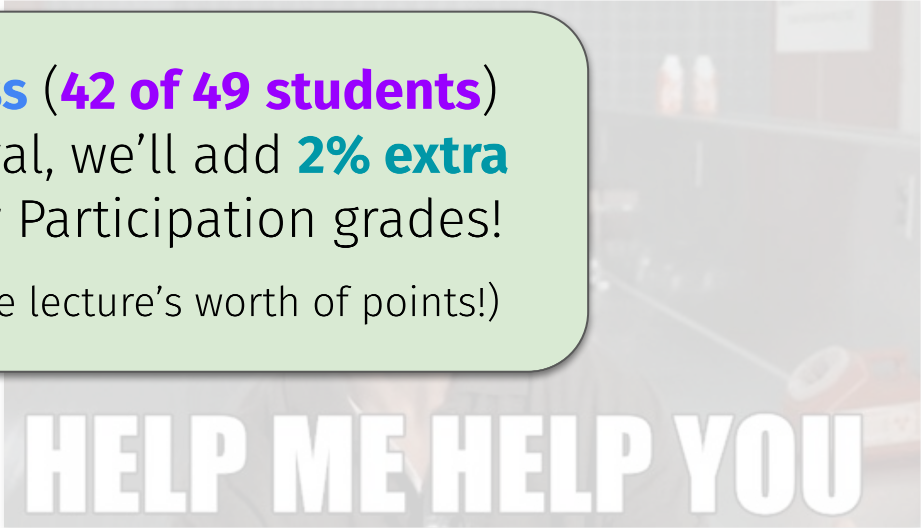
Project Title	**Fuzzing Resume-Matcher**
Team Members	Sunithya Penumarthy,Rahul Mandava,Harshita Samala,Leela Sowmya Jandhyala
Project Title	**Fuzzing Libqrencode**
Team Members	Vikas Kommalapati, Pranav Madhav, Kishore Kumar Kampalli, Anjali Kampalli
Project Title	**Fuzzing llama-cpp**
Team Members	Hao Ren, ChenCheng Mao, Ruiyang Xia
Project Title	**Fuzzing Eclipse Paho C client library**
Team Members	Nazmus Shakib Sayom, Jainta Paul
Project Title	**Fuzzing GUI Applications**
Team Members	Tanner Rowlett
Project Title	**Fuzzing Stormlib**
Team Members	Tinh Nguyen, Leo Leano, Austin Li, Vasil Vassilev
Project Title	**Fuzzing Open5G**
Team Members	Axe Tang
Project Title	**Trying to improve fuzzer coverage on Tesseract**
Team Members	Yash Lele, Rutuja Bhirud, Duncan Gilbreath
Project Title	**Fuzzing on struct_mapping library**
Team Members	Navya Dommalapati, Savant Mullapudi, Rajesh Vempalla, Shivahari Gundeti

End-of-semester Course Evals

- **I want your feedback!**
  - 3rd time teaching this course 😃
  - **Help me improve the class!**
- Due by **May 6th**
  - https://scf.utah.edu
  - **Please please please!**

End-of-semester Course Evals

- **I want your feedback!**
  - 3rd time teaching this course 😃
  - **Help me im**
- Due by **May 6**
  - https://scf.
  - **Please plea**



If 85% of class (**42 of 49 students**) submits an eval, we'll add **2% extra credit** to your Participation grades!

(equivalent to one lecture's worth of points!)

Questions?

Config-aware Fuzzing

Software Variability: a Tale of Two Perspectives

- Variability at **run-time**:
  - Different ways of parsing input
  - Current focus of fuzzing research

```
readelf: Warning: Nothing to do.
Usage: readelf <option(s)> elf-file(s)
 Display information about the contents of ELF format files
 Options are:
  -a --all               Equivalent to: -h -l -S -s -r -d -V -A -I
  -h --file-header       Display the ELF file header
  -l --program-headers   Display the program headers
     --segments          An alias for --program-headers
  -S --section-headers   Display the sections' header
     --sections          An alias for --section-headers
  -g --section-groups    Display the section groups
  -t --section-details   Display the section details
  -e --headers           Equivalent to: -h -l -S
  -s --syms              Display the symbol table
     --symbols           An alias for --syms
     --dyn-syms          Display the dynamic symbol table
     --lto-syms          Display LTO symbol tables
     --sym-base=[0|8|10|16]
                         Force base for symbol sizes.  The options are
                         mixed (the default), octal, decimal, hexadecimal.
```

Software Variability: a Tale of Two Perspectives

- Variability at **run-time**:
  - Different ways of parsing input
  - Current focus of fuzzing research

- Variability at **compile-time**:
  - Including & excluding certain code
  - Potentially huge attack surface
  - Not currently being explored

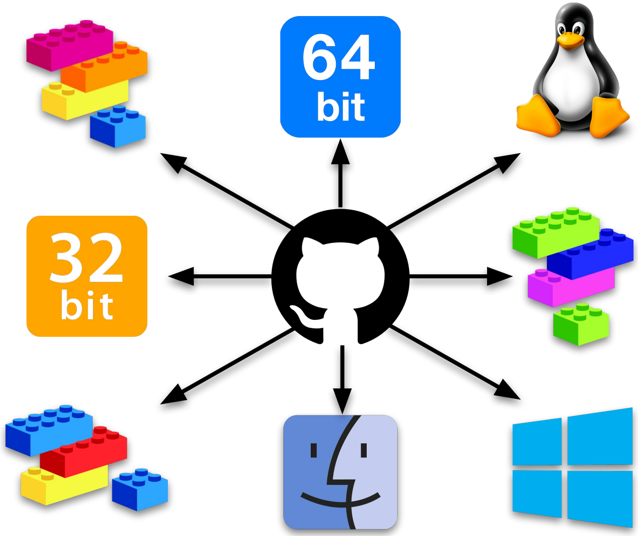
```
readelf: Warning: Nothing to do.
Usage: readelf <option(s)> elf-file(s)
 Display information about the contents of ELF format files
 Options are:
  -a --all               Equivalent to: -h -l -S -s -r -d -V -A -I
  -h --file-header       Display the ELF file header
  -l --program-headers   Display the program headers
     --segments          An alias for --program-headers
  -S --section-headers   Display the sections' header
     --sections          An alias for --section-headers
  -g --section-groups    Display the section groups
  -t --section-details   Display the section details
  -e --headers           Equivalent to: -h -l -S
  -s --syms              Display the symbol table
     --symbols           An alias for --syms
     --dyn-syms          Display the dynamic symbol table
     --lto-syms          Display LTO symbol tables
     --sym-base=[0|8|10|16]
                         Force base for symbol sizes.  The options are
                         mixed (the default), octal, decimal, hexadecimal.
```

Compile-time Variability

- Maintaining **deployment-specific** codebases is unscalable
  - Support on-demand features, environments, architectures

- **Solution:** software variability
  - One codebase, multiple builds

- Mechanisms for variability:
  - **C and C++:** the preprocessor
  - **Rust:** conditional compilation



Compile-time Variability Bugs

- Bugs triggerable only within a **specific variant** of the software

```
#ifdef TWL4030_CORE
int twl_probe()
{
  int *ops = NULL;

#ifdef OF_IRQ
  ops = &irq_domain_simple_ops;
#endif

  int irq = *ops;
}
#endif
```

http://vbdb.itu.dk/linux/6252547.html

Can you spot the bug?

If `TWL4030_CORE` and `!OF_IRQ`, then `int *ops` remains NULL…

= NULL pointer dereference!

With **thousands to millions** of possible variants, **concurrent** testing becomes **unscalable**!

Compile-time Variability Bugs

- http://vbdb.itu.dk/database.html

42 Variability Bugs in the Linux Kernel: A Qualitative Analysis

Iago Abal
iago@itu.dk

Claus Brabrand
brabrand@itu.dk

Andrzej Wąsowski
wasowski@itu.dk

IT University of Copenhagen
Rued Langgaards Vej 7, 2300 Copenhagen S, Denmark

ABSTRACT

Feature-sensitive verification pursues effective analysis of the exponentially many variants of a program family. However, researchers lack examples of concrete bugs induced by variability, occurring in real large-scale systems. Such a collection of bugs is a requirement for goal-oriented research, serving to evaluate tool implementations of feature-sensitive analyses by testing them on real bugs. We present a qualitative study of 42 variability bugs collected from bug-fixing commits to the Linux kernel repository. We analyze each of the bugs, and record the results in a database. In addition, we provide self-contained simplified C99 versions of the bugs, facilitating understanding and tool evaluation. Our study provides insights into the nature and occurrence of variability bugs in a large C software system, and shows in what ways variability affects and increases the complexity of software bugs.

Features in a configurable system interact in non-trivial ways, in order to influence each others functionality. When such interactions are unintended, they induce bugs that manifest themselves in certain configurations but not in others, or that manifest differently in different configurations. A bug in an individual configuration may be found by analyzers based on standard program analysis techniques. However, since the number of configurations is exponential in the number of features, it is not feasible to analyze each configuration separately.

Family-based [33] analyses, a form of feature-sensitive analyses, tackle this problem by considering all configurable program variants as a single unit of analysis, instead of analyzing the individual variants separately. In order to avoid duplication of effort, common parts are analyzed once and the analysis forks only at differences between variants. Re-

Is anyone fuzzing for compile-time variability bugs?

- **OSS-Fuzz**
  - x86
  - x64
  - ARM (maybe?)

```
export CFLAGS="$CFLAGS -DSQLITE_MAX_LENGTH=128000000 \
                -DSQLITE_MAX_SQL_LENGTH=128000000 \
                -DSQLITE_MAX_MEMORY=25000000 \
                -DSQLITE_PRINTF_PRECISION_LIMIT=1048576 \
                -DSQLITE_DEBUG=1 \
                -DSQLITE_MAX_PAGE_COUNT=16384"
```

- **SyzKaller**
  - Many kernels
  - Default configs only

stable-6.1-arm64-kasan-base.config
stable-6.1-arm64-kasan.config
stable-6.1-kasan-base.config
stable-6.1-kasan.config

Is anyone fuzzing for compile-time variability bugs?

- **OSS-Fuzz**
  - x86
  - x64
  - ARM (maybe?)

```
export CFLAGS="$CFLAGS -DSQLITE_MAX_LENGTH=128000000 \
               -DSQLITE_MAX_SQL_LENGTH=128000000 \
               -DSQLITE_MAX_MEMORY=25000000 \
               -DSQLITE_PRINTF_PRECISION_LIMIT=1048576 \
               -DSQLITE_DEBUG=1 \
               -DSQLITE_MAX_PAGE_COUNT=16384"
```

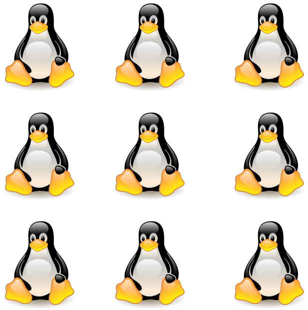
- **SyzKaller**
  - Many kernels
  - Default configs only

stable-6.1-arm64-kasan-base.config
stable-6.1-arm64-kasan.config
stable-6.1-kasan-base.config
stable-6.1-kasan.config

- An **under-explored** class of bugs
  - We need tools to find them!

What would compile-time variability fuzzing look like?

- **Crude** approach:
  - Enumerate every config possible
  - Concurrently fuzz all the builds
  - Differential execution



What would compile-time variability fuzzing look like?

- **Crude** approach:
  - Enumerate every config possible
  - Concurrently fuzz all the builds
  - Differential execution

- **Problem:** combinatorial explosion
  - Good luck building every config
  - Good luck fuzzing every build

Static Analysis of Variability in System Software: The 90,000 `#ifdefs` Issue*

System software can be configured at compile time to tailor it with respect to a broad range of supported hardware architectures and application domains. The Linux v3.2 kernel, for instance, provides more than 12,000 configurable features, which control the configuration-dependent inclusion of 31,000 source files with 89,000 `#ifdef` blocks.

Linux provides more than **12,000** configurable features

Toward Compile-time Variability-Aware Fuzzing

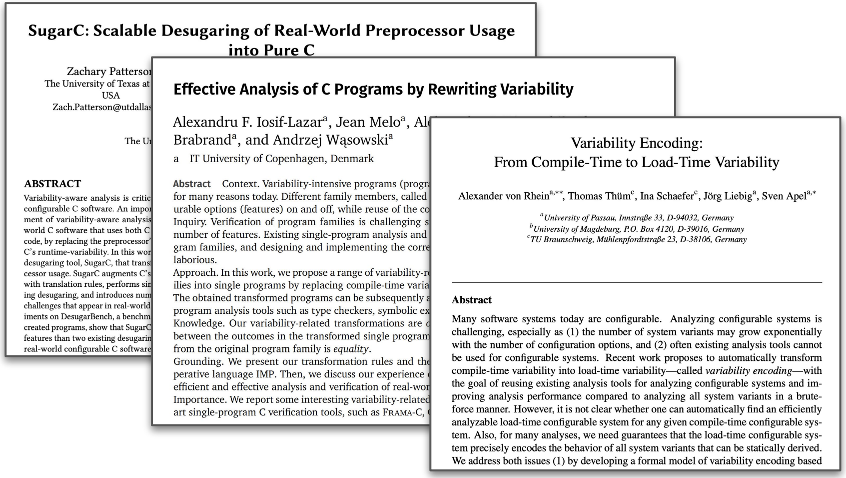
- **Idea:** transform **conditionally-compiled** code into **conditionally-invoked**

```c
#ifdef TWL4030_CORE
int twl_probe()
{
  int *ops = NULL;

#ifdef OF_IRQ
  ops = &irq_domain_simple_ops;
#endif

  int irq = *ops;
}
#endif
```

Prior work in the SE community on **"desugaring"** preprocessor usage:



SugarC: Scalable Desugaring of Real-World Preprocessor Usage into Pure C

Zachary Patterson
The University of Texas at
USA
Zach.Patterson@utdallas

ABSTRACT

Effective Analysis of C Programs by Rewriting Variability

Alexandru F. Iosif-Lazar[a], Jean Melo[a], Al
Brabrand[a], and Andrzej Wąsowski[a]

a IT University of Copenhagen, Denmark

Variability Encoding:
From Compile-Time to Load-Time Variability

Alexander von Rhein[a,**], Thomas Thüm[c], Ina Schaefer[c], Jörg Liebig[a], Sven Apel[a,*]

[a] University of Passau, Innstraße 33, D-94032, Germany
[b] University of Magdeburg, P.O. Box 4120, D-39016, Germany
[c] TU Braunschweig, Mühlenpfordtstraße 23, D-38106, Germany

Abstract

Many software systems today are configurable. Analyzing configurable systems is challenging, especially as (1) the number of system variants may grow exponentially with the number of configuration options, and (2) often existing analysis tools cannot be used for configurable systems. Recent work proposes to automatically transform compile-time variability into load-time variability—called *variability encoding*—with the goal of reusing existing analysis tools for analyzing configurable systems and improving analysis performance compared to analyzing all system variants in a brute-force manner. However, it is not clear whether one can automatically find an efficiently analyzable load-time configurable system for any given compile-time configurable system. Also, for many analyses, we need guarantees that the load-time configurable system precisely encodes the behavior of all system variants that can be statically derived. We address both issues (1) by developing a formal model of variability encoding based

Toward Compile-time Variability-Aware Fuzzing

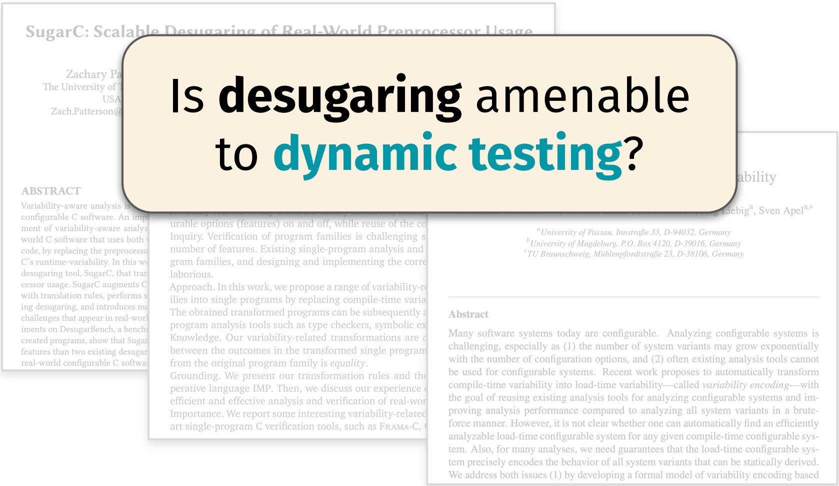
- **Idea:** transform **conditionally-compiled** code into **conditionally-invoked**

```
#ifdef TWL4030_CORE
int twl_probe()
{
  int *ops = NULL;

#ifdef OF_IRQ
  ops = &irq_domain_simple_ops;
#endif

  int irq = *ops;
}
#endif
```

Prior work in the SE community on "**desugaring**" preprocessor usage:



Is **desugaring** amenable to **dynamic testing**?

Toward Compile-time Variability-Aware Fuzzing

- **Idea:** transform **conditionally-compiled** code into **conditionally-invoked**

```
#ifdef TWL4030_CORE
int twl_probe()
{
  int *ops = NULL;

#ifdef OF_IRQ
  ops = &irq_domain_simple_ops;
#endif

  int irq = *ops;
}
#endif
```

Parse the AST to identify all preprocessor tokens

→

Transform all preprocessor code blocks

```
if (getenv("TWL4030_CORE"){
int twl_probe()
{
  int *ops = NULL;

if (getenv("OF_IRQ"){
  ops = &irq_domain_simple_ops;
}

  int irq = *ops;
}
}
```

Toward Compile-time Variability-Aware Fuzzing

- **Idea:** transform **conditionally-compiled** code into **conditionally-invoked**

```
if (getenv(“TWL4030_CORE”){
int twl_probe()
{
  int *ops = NULL;

if (getenv(“OF_IRQ”){
  ops = &irq_domain_simple_ops;
}

  int irq = *ops;
}
}
```

- Use fuzzing to **differentially test** different feature combinations

```
 TWL4030_CORE &&  OF_IRQ
!TWL4030_CORE && !OF_IRQ
!TWL4030_CORE &&  OF_IRQ
 TWL4030_CORE && !OF_IRQ
```

Toward Compile-time Variability-Aware Fuzzing

- **Idea:** transform **conditionally-compiled** code into **conditionally-invoked**

```
if (getenv("TWL4030_CORE")){
int twl_pro
{
  int *ops

if (getenv(
  ops = &irq_domain_simple_ops;
}

  int irq = *ops;
}
}
```

- Use fuzzing to **differentially test** ...tions

Goal: enable a **software product line** to be expressed and fuzzed **via a single executable**

```
                        _IRQ
 !TWL4030_CORE && !OF_IRQ
 !TWL4030_CORE &&  OF_IRQ
  TWL4030_CORE && !OF_IRQ
```

Current Work: Variability Desugaring

- **Current support for:**
  - `#ifdef`
  - Function decls/defs
  - Variable decls/defs
  - Nested macros
- **Working on:**
  - Other macro types
  - Duplicate-named vars/funcs
  - Other non-trivial cases

```
#ifdef FOO
void foo(){
  printf("foo");
}
#endif

int main(){
#ifdef FOO
int y = 2;
x = y;
foo();
#endif
}
```

```
void foo(){
  assert(“FOO”);
  printf(“foo”);
}

int main(){
  int y;
  if ("FOO"){
    y=2;
    x=y;
    foo();
  }
}
```

Current Work: Differential Execution

- **Initial approach:** replaying inputs
  - Gather high-coverage inputs
  - Replay each with instrumentation
  - Randomize enabled/disabled macros
  - Filter-out invalid configs pre/post fuzzing
  - Use standard bug oracles (e.g., ASAN)

- **Smarter approach:** track variability
  - Need to identify metrics of "interesting"
  - Prioritize configs that execute new paths
  - Prioritize configs that change program state

Future Directions

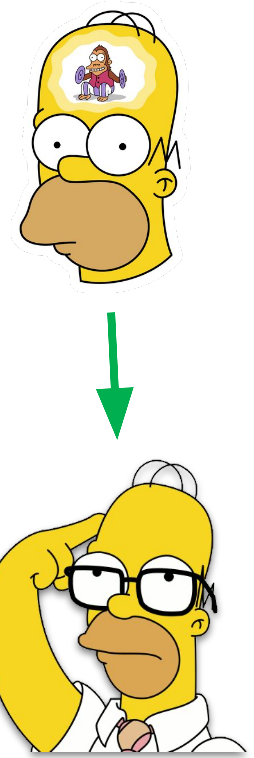
- AFL + SyzKaller implementations
  - Prototype a many-build approach
  - Eventually pair with desugaring
- Directed Variability Fuzzing
  - Pick subset of features to test
  - Constrain to specific path/region
- Cross-platform Variability
  - Test Windows and Linux builds
  - Explore other architectures

Course Wrap-up

You've finished the course!

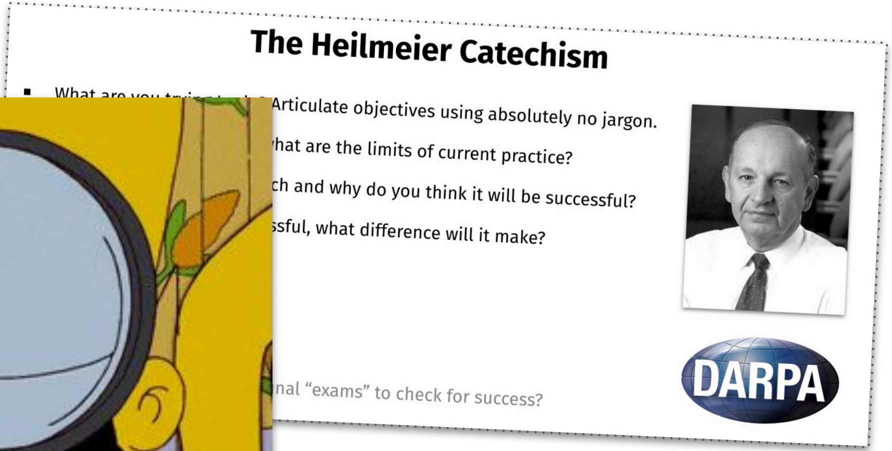
What did we learn?

- **Weeks 1 – 3:** Systems Research 101
  - Ideas, writing, presenting, reviewing
- **Weeks 4 – 9:** Fuzzing Fundamentals
  - Generation, feedback, bugs & triage, harnessing, roadblocks, fuzzing science
- **Weeks 10 – 12:** Emergent Enhancements
  - Optimization, directed fuzzing, hybrid fuzzing
- **Weeks 13 – 16:** New Frontiers in Fuzzing
  - Kernels, compilers, hardware, and more!



Goal #1: become better researchers

Part 1: Course Introduction	
Tuesday Meeting	Thursday Meeting
Aug. 23 **Course Introduction** (slides)	Aug. 25 **Research 101: Ideas** (s
Aug. 30 **Research 101: Writing** (slides)	Sep. 1 **Research 101: Review** (and a guest lecture by
Sep. 6 **Research 101: Presenting** (slides) **Select papers to present by 11:59pm**	Sep. 8 **Fuzzing Introduction** (s **Beginner Fuzzing Lab**



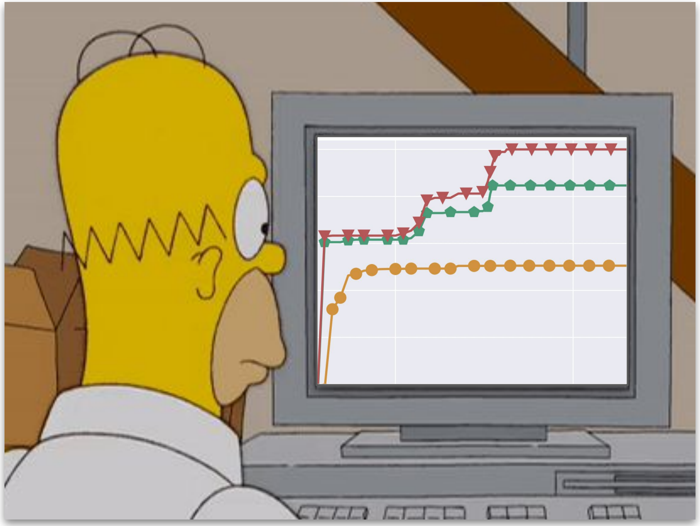
Feel free to bookmark or download the Research 101 slides!

cs.utah.edu/~snagy/courses/cs5963/slides

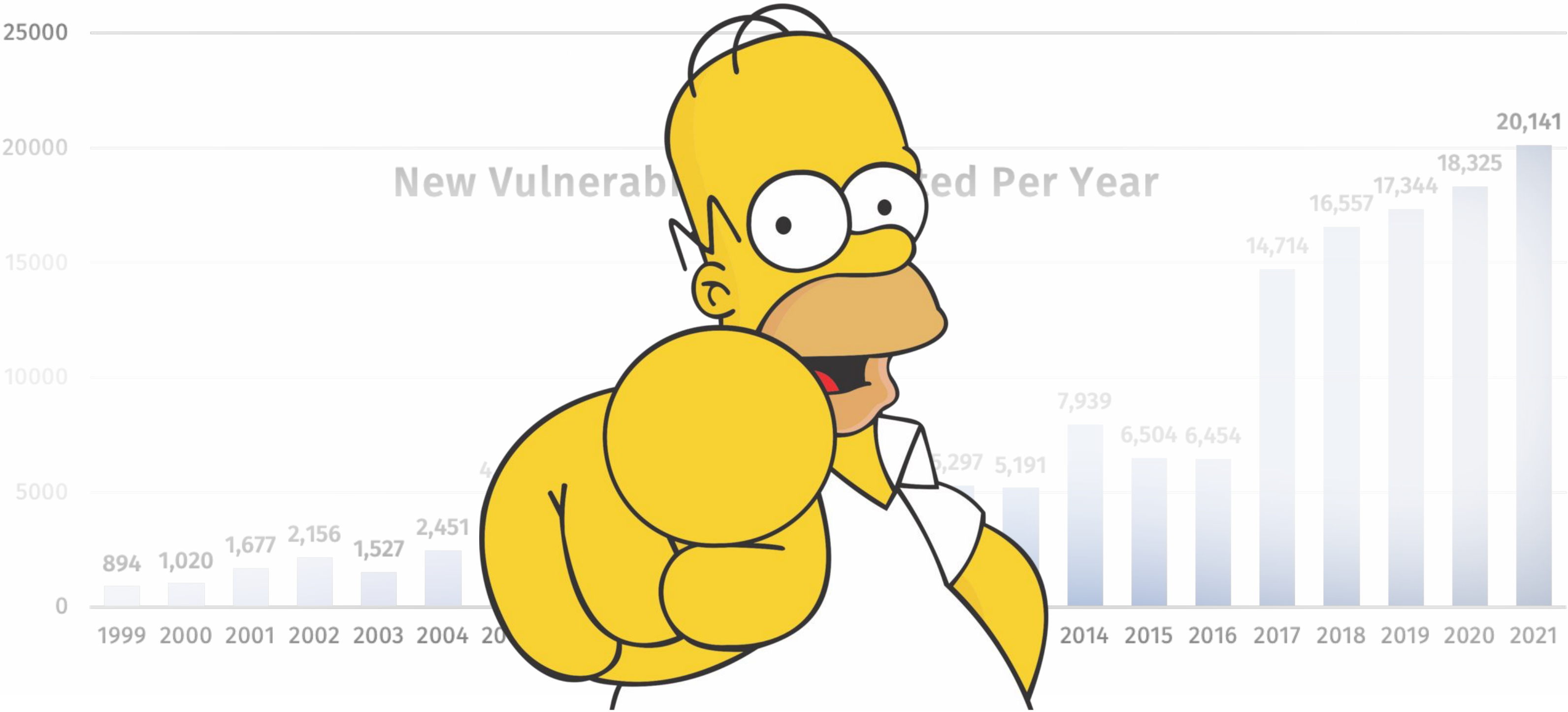
Goal #2: exposure to different perspectives

Goal #3: learn state-of-the-art tools

```
== ASAN: heap-use-after-free on address
0x61900000047f at pc 0x00000040a52c bp
0x7fff9200dbf0 sp 0x7fff9200dbe0
READ of size 1 at 0x61900000047f thread T0
    #0 0x40a52b in src/main.cpp:30
    #1 0x40e088 in std_function.h:297
    #2 0x40d605 in std_function.h:687
    #3 0x40b8d5 in src/
    #4 0x7f9a498ff412                   :308
```



Now go forth and teach others!



Thank you!