CSE 703 Seminar:
Advanced Software Security - Techniques and Tools

Instructor: Dr. Ziming Zhao

Location: Online
Time: Monday, 12:50 PM-2:55 PM

First off, Logistics!

Turn on camera if possible

Classes are recorded and can be accessed with an UB account

Webpage: https://zzm7000.github.io/teaching/2021springcse703/index.html

Feel free to interrupt me and ask questions

Eat or drink if you need

Instructor

Dr. Ziming Zhao
Assistant Professor, CSE
Director, CyberspAce seCuriTy and forensIcs Lab (CactiLab)

Email: zimingzh@buffalo.edu
http://zzm7000.github.io
http://cactilab.github.io

Office: 338B Davis Hall / Online
Office hours: By appointment

Agenda

1. Course overview
2. How to do cybersecurity research?
3. How to do software and system security research?
4. Topic 1: Reverse engineering

Course Overview

Course Goals

- Introduce you to cybersecurity research.
- Introduce you to software security research.
- Teach you how to read papers.
- Teach you how to give good paper presentations.
- Teach you how to write good paper reviews.

Course Format

Lecture by the instructor

- Background knowledge
- Demonstrate tools

Paper presentations

- One student will be presenting and leading the discussion.

Course Topics and Class Schedule

On the website

Also take a look at
https://zzm7000.github.io/teaching/2020fallcse610/index.html

Paper Presentations

Each student will present 2 papers from the reading list. To better prepare for the presentation, you are required to do the following:

1. Email me your answers to the talk preparation questions 3 days before the presentation.
2. Email me your slides 3 days before the presentation.

30 mins presentations
15 mins discussions

Paper Reviews

You will write reviews for 2/3 papers in the “Paper Presentations” column (you will choose which ones). The format of a review can be found here.

You cannot review the paper which you will present.

You have to submit a review before that paper is presented; Please only txt files; 600 hundred words at least; A template will be provided.

Reading/discussing papers in groups is highly encouraged but reviews have to be written individually. You are not allowed to use any online material for a review. A review may receive a zero grade due to plagiarism.

Class Discussions

Discussions are an important part of the course. You are expected to attend every class and ask questions/make comments.

Grading

1 credit

- Paper presentation: 30%
- Paper reviews (2): 30%
- Homework: 25%
- Class participation: 15%

3 credits

- Paper presentation: 30%
- Paper reviews (3): 25%
- Class participation: 15%
- Homework: 15%
- Course project and presentation: 15%

Note that the final grade is S/U. To receive an S grade, you need to score 70% or more.

Course Project

1. MITRE eCTF - up to 6 people
    a. Join the weekly meeting at 3pm today if you want to participate

2. Analyze a smart meter - up to 4 people
    a. I will give you two smart meters
    b. Come up with a plan and milestones - in the first 3 weeks

3. Name your own project - the rest
    a. Come up with a plan and milestones - in the first 3 weeks

Let me know which option you choose by end of this week.

What is a capture-the-flag competition?

Two major software CTF formats

Jeopardy CTFs: They revolve around a set of challenges which are provided by competition organizers to competitors. Each challenge is designed so that when the competitor solves it, a small piece of text or "flag" is revealed. The flag is then submitted to a website or scoring engine in exchange for points. Competitors usually receive about 72 hours (typically the course of a weekend) to solve as many challenges as possible.

Attack & Defense CTFs: Teams are each given the same set of vulnerable server software. Teams are to setup & audit this software before the competition. At the start of the competition, teams will connect their servers to an isolated network to join the CTF. Within this network, teams will launch attacks against each others servers hoping to exploit the vulnerabilities they've found. Likewise, teams will need to properly patch their software so that it is protected against these exploits and functions normally.

babymem

level1_teaching1 1	level1_testing1 1	level1_testing2 1	level2_teaching1 1
level2_testing1 1	level2_testing2 1	level3_teaching1 1	level3_testing1 1
level3_testing2 1	level4_teaching1 1	level4_testing1 1	level4_testing2 1
level5_teaching1 1	level5_testing1 1	level5_testing2 1	level6_teaching1 1
level6_testing1 1	level6_testing2 1	level7_teaching1 1	level7_testing1 1
level7_testing2 1	level8_teaching1 1	level8_testing1 1	level8_testing2 1
level9_teaching1 1	level9_testing1 1	level9_testing2 1	level10_teaching1 1
level10_testing1 1	level10_testing2 1	level11_teaching1 1	level11_testing1 1
level11_testing2 1	level12_teaching1 1	level12_testing1 1	level12_testing2 1

How is eCTF different from other CTFs?

Offline, semester-long hardware/software CTF

How does it work?

This event puts competitors through the exercise of trying to create a secure system and then learning from their mistakes. The main target is a real physical embedded device, opening the challenge to include physical/proximal access attacks.

Secure Design

Teams design a secure system that meets all the challenge requirements.

Handoff

MITRE verifies that each submitted system has met all functional requirements.
MITRE posts designs for all teams to evaluate during the attack phase.

Attack

Teams perform security evaluations of opposing teams' systems and request provisioned chips for vulnerable systems. Points are awarded for flags retrieved from successful attacks.

Academic Integrity

- Discussion is encourage. But, you cannot share your code, exploits to your classmates or post them online.

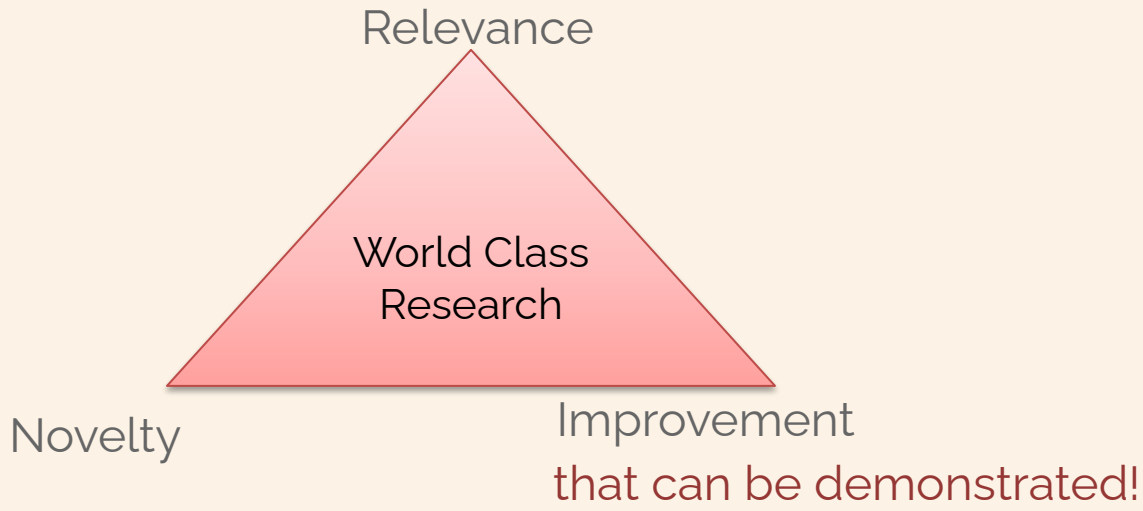
- The university, college, and department policies against academic dishonesty will be strictly enforced. To understand your responsibilities as a student read: UB Student Code of Conduct.

- Plagiarism or any form of cheating in homework, assignments, labs, or exams is subject to serious academic penalty.

- Any violation of the academic integrity policy will result in a 0 on the homework, lab or assignment, and even an **F** or **>F<** on the final grade. And, the violation will be reported to the Dean's office.

How to do cybersecurity research?

Properties of Good CS Research



Relevance

- Find a good problem to work on!
- What is a good problem?
  - We are NOT mathematicians; we do NOT even work on hard science
  - Many people care about the problem
  - Many people are affected by the problem
  - Program is broad or general
    - Big difference from industry security “researchers”: bug in specific application is typically not a good research problem
  - Problem has not been solved yet
    - Previously unknown problem
    - Existing solution have (severe) limitations
  - Solution should not be trivial. Well, if you can prove a trivial solution is perfect, that is also good.

Novelty

- Your solution/approach must be novel
- Optimally, you invent a completely novel technique
  - This does not happen often
- Apply a variant of existing technique (e.g., machine learning, static analysis)
  - First to apply technique to particular problem at hand
  - Interesting twist or extension (actually applying a theoretical solution in practice requires tweaks)
  - Try to generalize the techniques if possible
- Critical requirement to determine novelty
  - You must know related work very well!

How to Find Good Security Problems

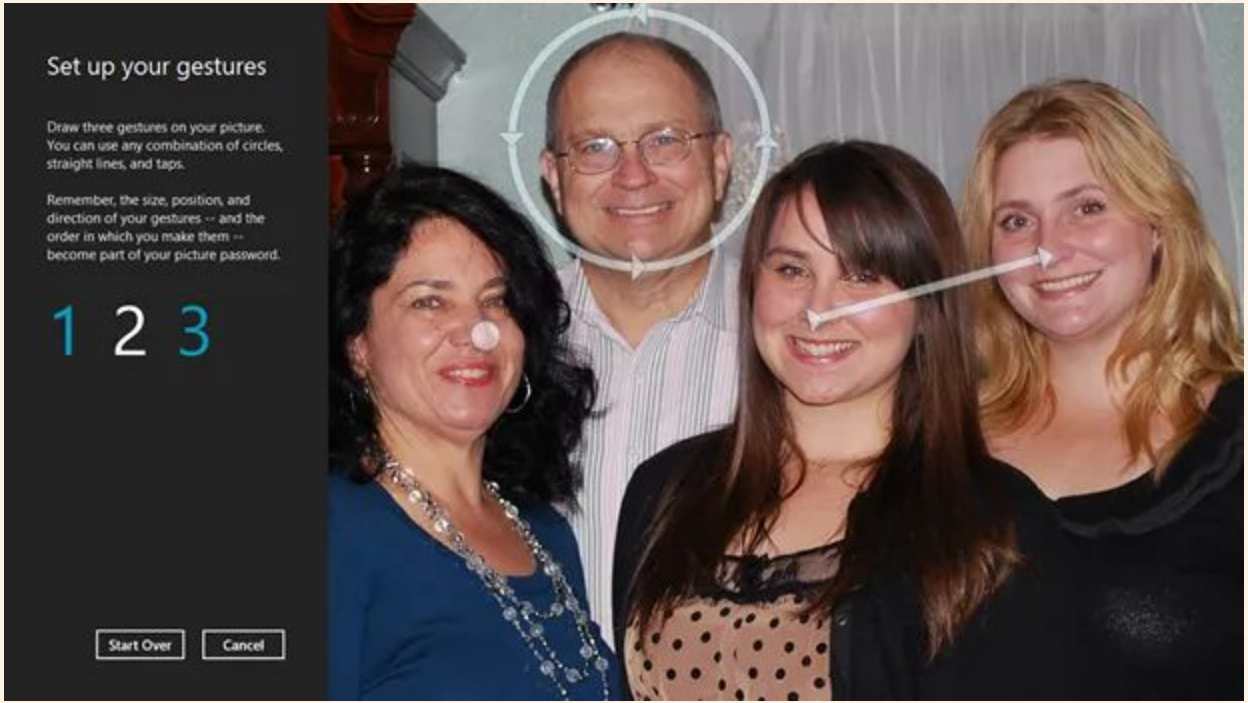
- Pay attention to your daily life!! All kinds of security problems.

- Dig into the details – Get your hand dirty
  - Read code, analyze recently published exploits, write code, experiment (System/software/network security)
  - Analyze traffic (Network security)
  - Analyze posts (Cybercrime analysis)

- Be **excited** when you find something new and difficult. Talk with me when you cannot solve the problem.

How to Find Good Security Problems

- My 1st USENIX Security Paper



How to Find Good Security Problems

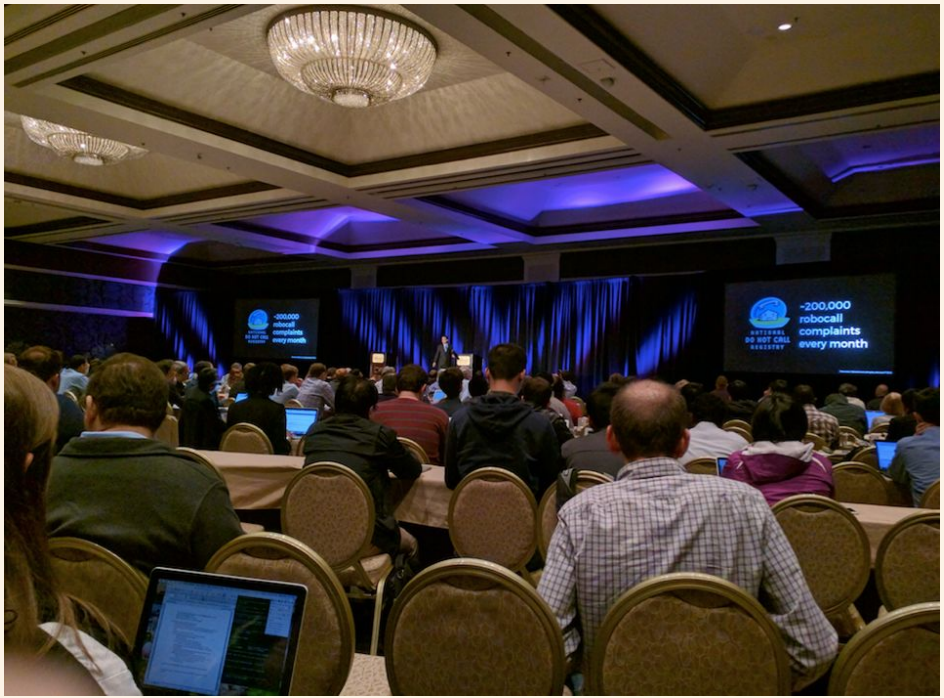
- My 1st USENIX Security Paper

How to Find Good Security Problems

- My 1st S&P Paper

How to Find Good Security Problems

- My 1st S&P Paper



Top Conferences in CS

- http://csrankings.org/#/index?all&us

Most Prestigious Security Venues

Rank 1	S&P (Oakland)	IEEE Symposium on Security and Privacy
	CCS	ACM Conference on Computer and Communications Security
	Crypto	International Cryptology Conference
	Eurocrypt	European Cryptology Conference
	Security	Usenix Security Symposium
	NDSS	ISOC Network and Distributed System Security Symposium
Rank 2	ESORICS	European Symposium on Research in Computer Security
	RAID	International Symposium on Recent Advances in Intrusion Detection
	ACSAC	Annual Computer Security Applications Conference
	DSN	The International Conference on Dependable Systems and Networks
	CSF (CSFW)	IEEE Computer Security Foundations Symposium. Supersedes CSFW (Computer Security Foundations Workshop)
	TCC	Theory of Cryptography Conference
	Asiacrypt	International Conference on the Theory and Application of Cryptology and Information Security
	IMC	Internet Measurement Conference
	PETS	Privacy Enhancing Technologies Symposium

http://faculty.cs.tamu.edu/guofei/sec_conf_stat.htm

Rank 3	SecureComm	Internation Conference on Security and Privacy in Communication Networks
	CNS	IEEE Conference on Communications and Network Security
	DIMVA	GI SIG SIDAR Conference on Detection of Intrusions and Malware and Vulnerability Assessment
	AsiaCCS	ACM Symposium on Information, Computer and Communications Security
	ACNS	International Conference on Applied Cryptography and Network Security
	FC	International Conference on Financial Cryptography and Data Security
	SAC	ACM Symposium on Applied Computing
	ACISP	Australasia Conference on Information Security and Privacy
	ICICS	International Conference on Information and Communications Security
	ISC	Information Security Conference
	ICISC	International Conference on Information Security and Cryptology
	SACMAT	ACM Symposium on Access Control Models and Technologies
	CT-RSA	RSA Conference, Cryptographers' Track
	IFIP SEC	IFIP International Information Security Conference
	WiSec (WiSe, SASN)	ACM Conference on Wireless Network Security Supersedes WiSe (ACM Workshop on Wireless Security) and SASN (ACM Workshop on Security of Ad-Hoc and Sensor Networks)
	SOUPS	Symposium On Usable Privacy and Security
	IFIP WG 11.9	IFIP WG 11.9 International Conference on Digital Forensics
	DFRWS	Digital Forensic Research Conference
	CODASPY	ACM Conference on Data and Application Security and Privacy
	MALWARE	International Conference on Malicious and Unwanted Software
	-- Workshops below --	
	FSE	Fast Software Encryption workshop
	PKC	International Workshop on Public-Key Cryptography
	NSPW	New Security Paradigms Workshop
	IH	Workshop on Information Hiding
	WSPEC	Workshop on Security and Privacy in E-commerce
	DRM	ACM Workshop on Digital Rights Management

Tracking People

- Remember recurring names and group for related papers
  - Take a moment after reading a paper to understand the paper's context
    - Who is the lead author of the paper?
    - Who is the professor/advisor on the paper (typically last author)?
- Read names of program committees
- Visit people's web pages (this is why having a web page is critical)
- Follow people who work on related projects
- Security Circus (http://s3.eurecom.fr/~balzarot/notes/top4_2019/, http://csrankings.org/)

Paper Reading

For those of you who are not used to reading research papers, I recommend reading "How to Read a Paper" by S. Keshav.

Critical Thinking - Reading

- Read a paper and consider:
  - Do I like it? Hate it? (opinion)
  - What problem is it trying to solve?
  - How does their approach differ from previous ones?
  - (how much previous work do I know about – read it! (reference chaining))
  - Does it work?
  - What could be improved?

Critical Thinking - Reading

- Consider a paper (or your thesis) as an argument; a paper is not a textbook
  - What is the problem?
  - If not well known, why is it a problem?
  - Why are all previous approaches insufficient (broken / wrong / stupid)?
  - What is your approach?
    - how does it work?
    - how well does it work?
    - how does it improve on previous attempts?

Critical Thinking - **Advanced Reading**

- Read the abstract/intro a paper and consider:
  - What problem is it trying to solve?
  - How do I solve this problem without reading their solutions?
  - Compare your approach with their approach.
  - Is my approach significantly better? Yes: Congratulations, you have a research idea right away!

Critical Thinking - **Advanced Reading**

- Learn their thinking process
- Learn their approach to build up your own skill set
- Learn their paper writing

- Do not directly follow their topic
  - If they already published a top conference paper in this topic, it means they may already solve the problem

Critical Thinking - **Advanced Reading**

Reading papers should be your new hobby. Make sure you read everyday!

Level of Paper Understanding

- Level 1. Understand what the problem is; understand what the authors tried to do
- Level 2. Understand the high-level idea of the proposed approach to solve the problem
- Level 3. Can run and use the code/tool from this project
- Level 4. Understand the source code of this project
- Level 5. Can replicate this project by yourself [You may not want to do that if code is available]

How to do world-class system and software security research?

What is System and Software Security Research?

- Work in hardware, OS, file systems, databases, networking, compiler, language run-times, …
- Not a ‘hard’ science; a lot of **engineering**
  - No ground truth to be discovered
    - Get to create the universe!
  - Things can be “sort of” right
    - (Engineering) Building interesting systems
    - Absolutes are rare
- Key skill: ***critical thinking***, ***hands-on ability (debug)***

Solid Background Needed in ...

Hardware Architecture: CPU, Controllers (Cortex-M, Cortex-A, RISC-V, etc.)

Operating System Design and Implementation (Linux, xv6, Mbed, ARM-TF, seL4, etc.)

Compiler Design and Implementation (LLVM, etc.)

Program Analysis on C/C++ and Binary (Symbolic Execution, Compiler back-end, LLVM, Angr, fuzzing, etc.)

Hacking, Reverse Engineering (Ghidra, Angr, IDA Pro, etc.)

Topic 1: x86/x64 Binary Disassembly

SoK: All You Ever Wanted to Know About x86/x64 Binary Disassembly But Were Afraid to Ask

Chengbin Pang*‡§ Ruotong Yu* Yaohui Chen† Eric Koskinen* Georgios Portokalidis* Bing Mao‡ Jun Xu*

*Stevens Institute of Technology †Facebook Inc. ‡Nanjing University

***Abstract*—Disassembly of binary code is hard, but necessary for improving the security of binary software. Over the past few decades, research in binary disassembly has produced many tools and frameworks, which have been made available to researchers and security professionals. These tools employ a variety of strategies that grant them different characteristics. The lack of systematization, however, impedes new research in the area and makes selecting the right tool hard, as we do not understand the strengths and weaknesses of existing tools. In this paper, we systematize binary disassembly through the study of nine popular, open-source tools. We couple the manual examination of their code bases with the most comprehensive experimental evaluation (thus far) using 3,788 binaries. Our study yields a comprehensive description and organization of strategies for disassembly, classifying them as either *algorithm* or else *heuristic*. Meanwhile, we measure and report the impact**

TABLE I: The group of open-source tools that our study covers and representative works that use those tools.

Tool (Version)	*Source (Release Date)*	*Public Use*
PSI (1.0)	Website [63] (Sep 2014)	[50, 88, 111]
UROBOROS (0.11)	Github [93] (Nov. 2016)	[103]
DYNINST (9.3.2)	Github [79] (April 2017)	[7, 18, 69, 73, 96]
OBJDUMP (2.30)	GNU [47] (Jan. 2018)	[21, 103, 111]
GHIDRA (9.0.4)	Github [75] (May 2019)	[24, 45, 91]
MCSEMA (2.0.0)	Github [13] (Jun. 2019)	[22, 41, 44]
ANGR (8.19.7.25)	Github [8] (Oct. 2019)	[20, 71, 81, 98, 112]
BAP (2.1.0)	Github [26] (Mar. 2020)	[10, 16, 64]
RADARE2 (4.4.0)	Github [89] (April 2020)	[4, 31, 52, 58]

- **Algorithms** typically produce results with some correctness guarantees. They mostly leverage knowledge from the binary (*e.g.,* symbols), the machine (*e.g.,* instruction set),

Oakland 2021

What is Binary Disassembly?

Disassembly is the process of recovering the assembly instructions of a binary.

Symbolization determines cross-references (xrefs for short) or precisely, numeric values in the binary that are references of other code or data objects. Depending on the location of the reference and the location of the target, there are four types of xrefs: code-to-code (c2c), code-to-data (c2d), data-to-code (d2c), and data-to-data (d2d).

Function Entry Identification locates the entry points of functions. A special but important case is the main function.

CFG Reconstruction re-builds the control flow graph (CFG) of a binary program. We consider direct control transfers, indirect jumps/calls, tail calls, and non-returning functions.

Why is Binary Disassembly Difficult?

Disassembly is the process of recovering the assembly instructions of a binary.

Correctly disassembling a binary is **challenging**, mainly owing to the loss of information (e.g., symbols and types) occurring when compiling a program to machine code and the complexity of constructs (e.g., jump tables, data embedded in code, etc.) used to efficiently implement language features.

Automatic/Interactive Disassembly Tools

Objdump
GDB
IDA Pro
Ghidra
Binary Ninja - web

Algorithms and Heuristics in Disassembly

Linear Sweep [OBJDUMP, PSI, UROBOROS]: Linear sweep continuously scans pre-selected code ranges and identifies valid instructions, exploiting the rationale that modern assemblers tend to layout code successively to reduce the binary's size.

In general, a linear sweep strategy can be described by how it selects sweep ranges and how it handles errors during scanning. As such, we summarize algorithms according to these two aspects.

Errors due to data-in-code

Algorithms and Heuristics in Disassembly

All tools in this class follow OBJDUMP to select code regions for sweep: they process code ranges specified by symbols in the **.symtab** and **.dynsym**.

OBJDUMP deems invalid opcodes as errors, **skips a byte**, and resumes scanning.

Beyond invalid opcodes, PSI considers control transfers to non-instructions as errors.

Algorithms and Heuristics in Disassembly

Recursive Descent [DYNINST, GHIDRA, ANGR, BAP, RADARE2]: Recursive descent starts with a given code address and performs disassembly following the control flow.

Strategies in this category usually consist of three components: (1) how to select code addresses, (2) how to resolve control flow, and (3) how to handle the code gaps left by recursive disassembly.

All the tools consider the program entry and available symbols as code addresses for recursive disassembly.

Algorithms and Heuristics in Disassembly

When encountering direct control transfers, the tools expand the disassembly to the targets. However, to handle indirect control transfers, different tools adopt different approaches.

Algorithms and Heuristics in Symbolization

Symbolization identifies numerical values in the binary that are actually references to code or data objects.

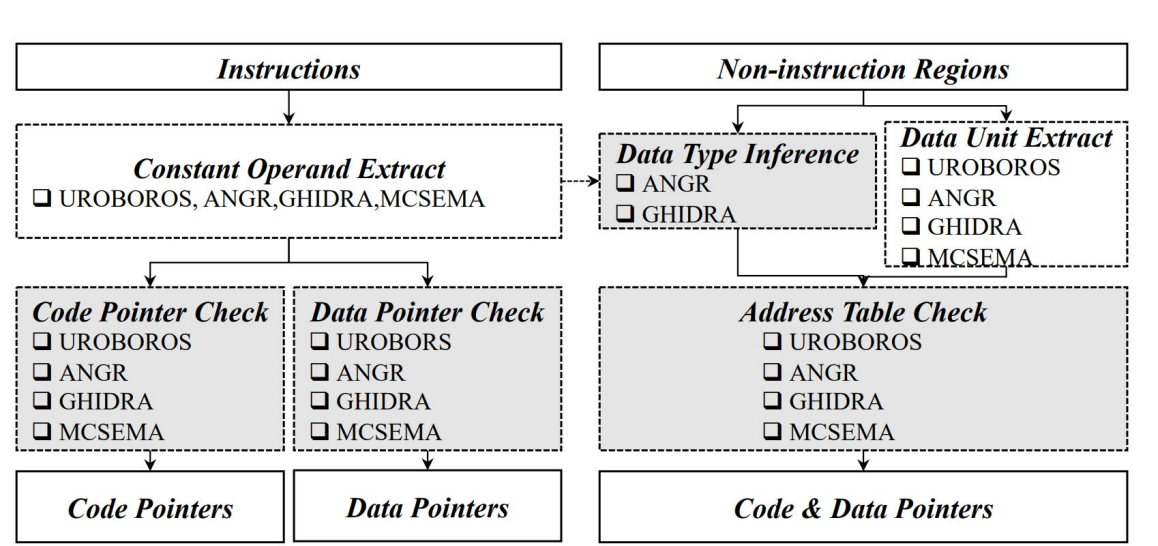


Fig. 2: A general workflow of symbolization.

Algorithms and Heuristics in Function Entry Identification

Main Function [DYNINST, ANGR, BAP, RADARE2]: To locate it, ANGR and BAP analyze the _start function and, following calling conventions, infer the first argument passed by _start to __libc_start_main.

```
1 48 c7 c7 e2 e0 40 00 mov $0x40e0e2,%rdi ;main
2 ff 15 ce 48 05 00 ** callq __libc_start_main
```

Listing 1: Call to `__libc_start_main` in `_start`.

Algorithms and Heuristics in Function Entry Identification

General Functions [DYNINST, GHIDRA, ANGR, BAP, RADARE2]: To identify the entries of non-main functions, these tools adopt a hybrid approach that consists of three parts: (1) The tools seek symbols remaining in the .symtab and .dynsym sections to determine known-to-be-good functions.

(2) All tools consider targets of direct calls to be function entries, while ANGR and GHIDRA additionally resolve certain indirect calls to determine more function entries. Finally, DYNINST, GHIDRA, ANGR and RADARE2 include targets of tail calls as function entries.

(3) All tools use pattern-based approaches to further recover functions. GHIDRA, ANGR and RADARE2 find function entries based on common prologues (or epilogues);

Algorithms and Heuristics in resolving indirect jumps/calls

Indirect Jumps [DYNINST, GHIDRA, ANGR, RADARE2]: Three types of indirect jumps:

(1) jump tables (compiled from switch-case and if-else statements);

(2) indirect tail calls (indirect calls optimized as tail calls); and

(3) handwritten ones (e.g., longjmp and other cases in Glibc [48]).

ANGR, given an indirect jump, considers the operand as a source and runs backwards slicing. In the sliced area, ANGR uses full-scale VSA to identify possible targets.

Homework

1. Read “How to read a paper”
2. Read “How to Review a Technical Paper”
3. Read “Review Guidelines”
4. Read “Presentation Guidelines”
5. Read “Sok: All you ever wanted to know about x86/x64 binary disassembly but were afraid to ask”