Week 12: Lecture A

Software Reverse Engineering

Tuesday, November 12, 2024

Announcements

- **Project 3** grades are now available on **Canvas**
- **Statistics:**
  - Average score: **97%**
  - Last year's avg: **90%**
- **Fantastic job!**

Announcements

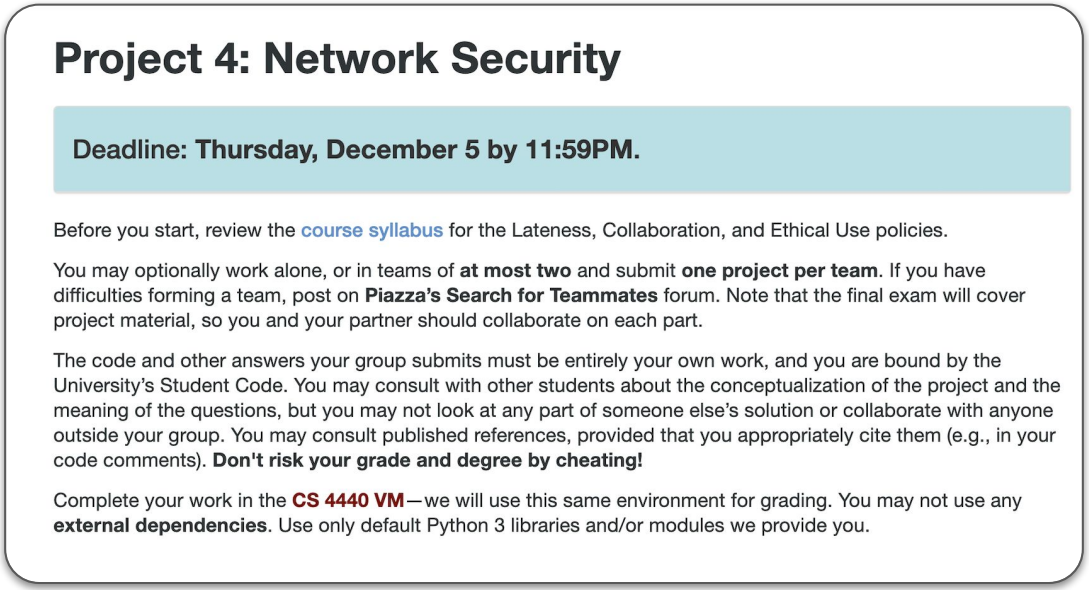
- **Project 3** grades are now available on **Canvas**

- Think we made an error? Request a regrade!
  - Valid regrade requests:
    - You have verified your solution is correct (i.e., we made an error in grading)

Project 3 Regrade Requests (see **Piazza** pinned link):
Submit by **11:59 PM** on **Monday 11/18** via **Google Form**

Announcements

- **Project 4: NetSec** released
  - **Deadline:** Thursday, December 5th by 11:59PM



Project 4: Network Security

Deadline: Thursday, December 5 by 11:59PM.

Before you start, review the course syllabus for the Lateness, Collaboration, and Ethical Use policies.

You may optionally work alone, or in teams of **at most two** and submit **one project per team**. If you have difficulties forming a team, post on **Piazza's Search for Teammates** forum. Note that the final exam will cover project material, so you and your partner should collaborate on each part.

The code and other answers your group submits must be entirely your own work, and you are bound by the University's Student Code. You may consult with other students about the conceptualization of the project and the meaning of the questions, but you may not look at any part of someone else's solution or collaborate with anyone outside your group. You may consult published references, provided that you appropriately cite them (e.g., in your code comments). **Don't risk your grade and degree by cheating!**

Complete your work in the **CS 4440 VM**—we will use this same environment for grading. You may not use any **external dependencies**. Use only default Python 3 libraries and/or modules we provide you.

Project 4 Progress

Working on Part 1.0: Password Cracking

0%

Working on Part 1.1: Port Scanning

0%

Working on Part 1.2: Anomalous Activity

0%

Finished Part 1, working on Part 2!

0%

None of the above

0%

Announcements

- I'll be out of town rest of week 💍
  - **No office hours on Thursday**
- **Guest lecturer** this Thursday
  - **Dr. Guanhong Tao:** attacking LLMs
  - Jailbreaking and backdoor attacks
- No PollEverywhere… but show up!
  - Final exam is still comprehensive 😉

https://tao.aisec.world/

Announcements

utahsec

See Discord for meeting info!

utahsec.cs.utah.edu

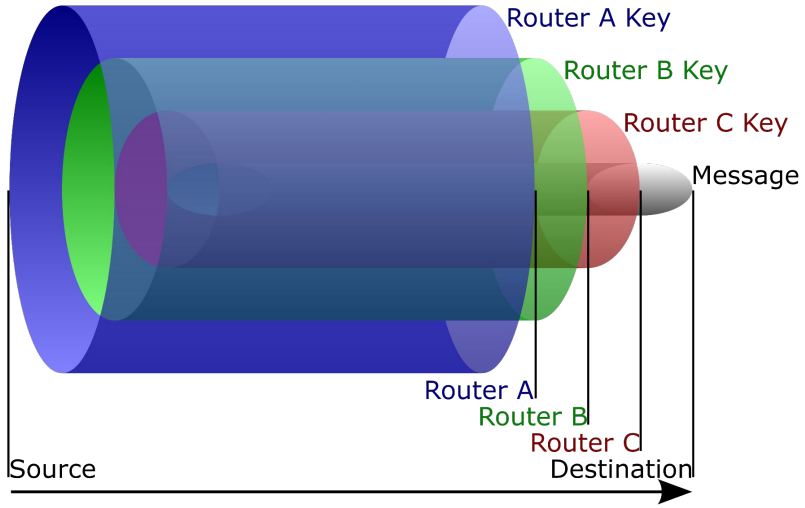
Questions?

Last time on CS 4440...

Security in Practice:
Tor—The Onion Router

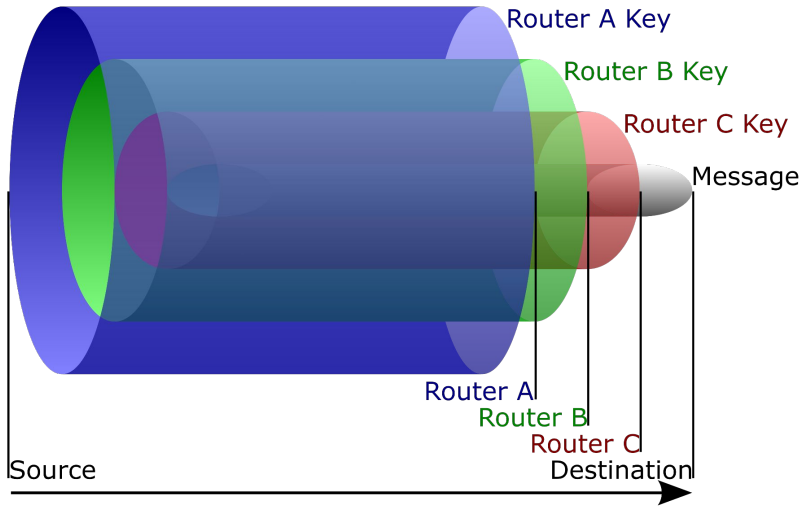
Anonymity Primitive: Onion Routing

- Each message is ???



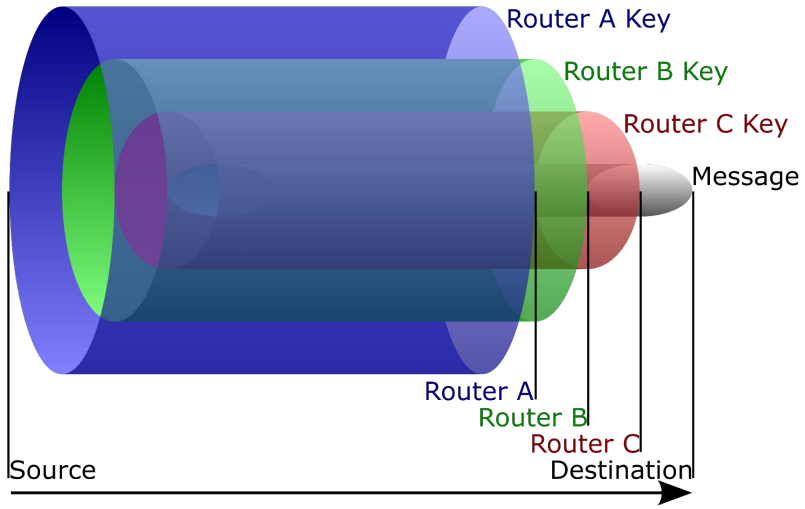
Anonymity Primitive: Onion Routing

- Each message is **repeatedly encrypted**
  - **Analogy:** multiple layers of an onion
- Sent through **multiple network nodes**
  - These nodes are called **onion routers**
  - Each node removes an encryption layer to uncover the message **routing instructions**
  - Process repeats when sent to next router
- **Anonymity:** prevents **???**

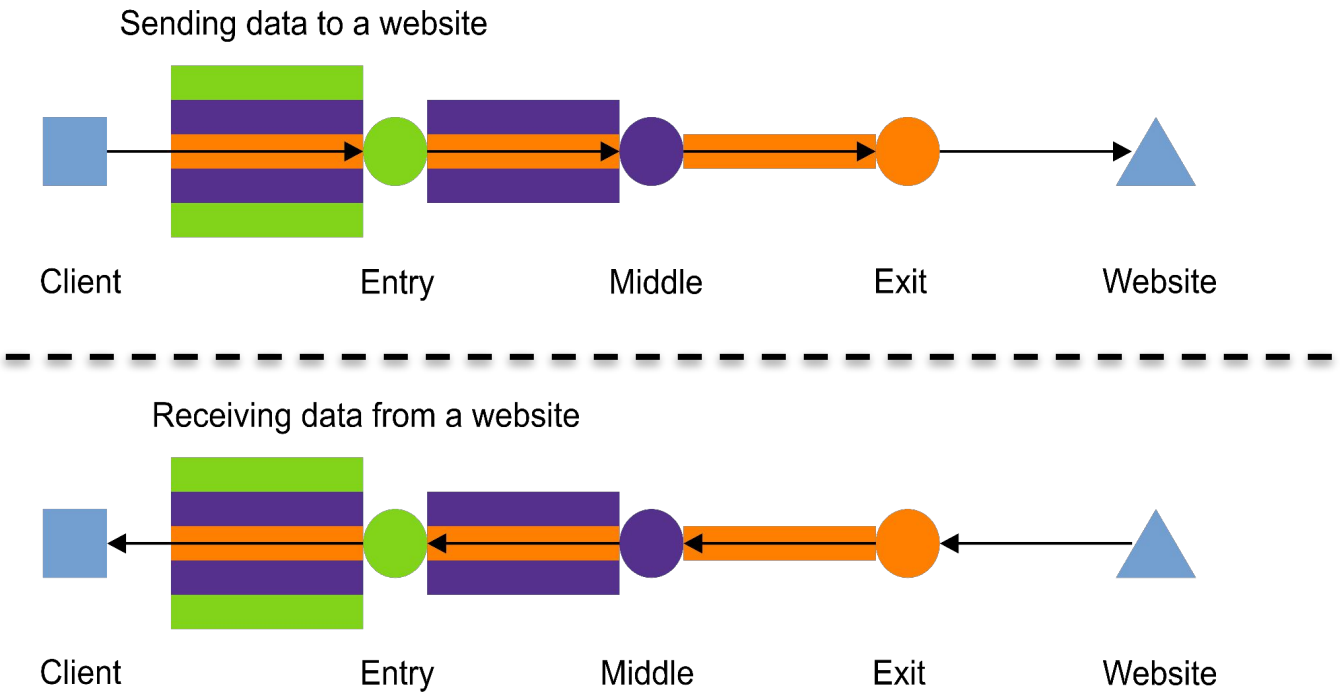


Anonymity Primitive: Onion Routing

- Each message is **repeatedly encrypted**
  - **Analogy:** multiple layers of an onion
- Sent through **multiple network nodes**
  - These nodes are called **onion routers**
  - Each node removes an encryption layer to uncover the message **routing instructions**
  - Process repeats when sent to next router
- **Anonymity:** prevents any intermediary nodes from knowing message **origin**, **destination**, and **contents**

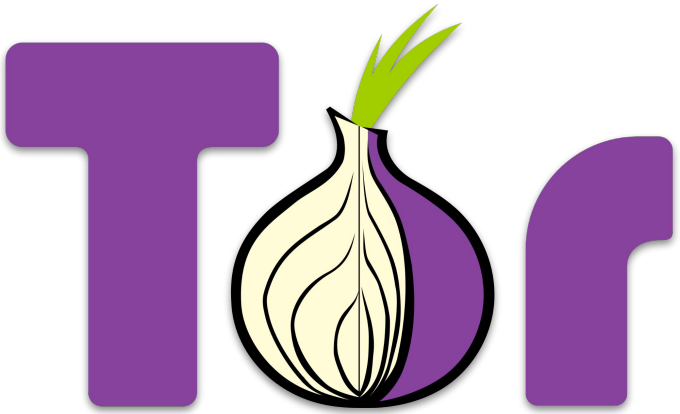


Onion Routing Visualized



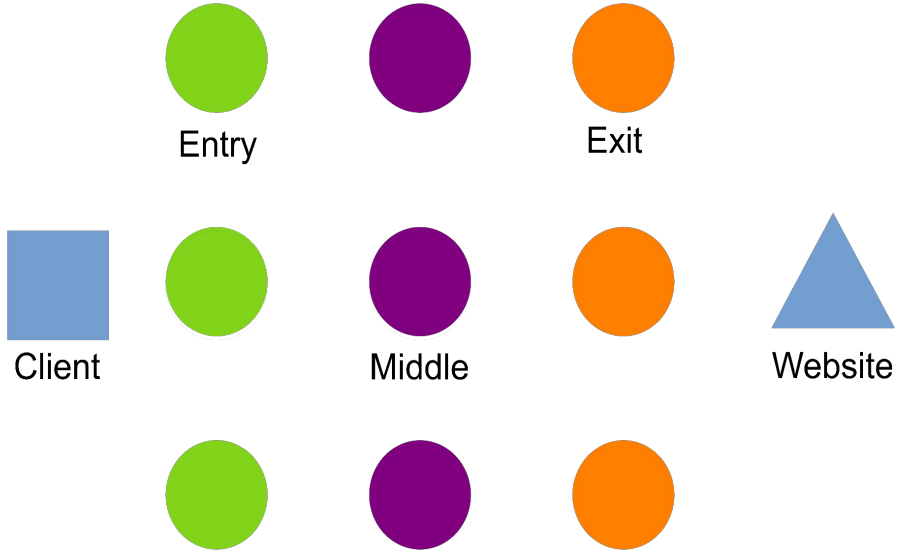
Tor: The Onion Router

- **Tor:** a distributed overlay network
  - Anonymizes TCP-based applications
    - Secure shell
    - Web browsing
    - Instant messaging



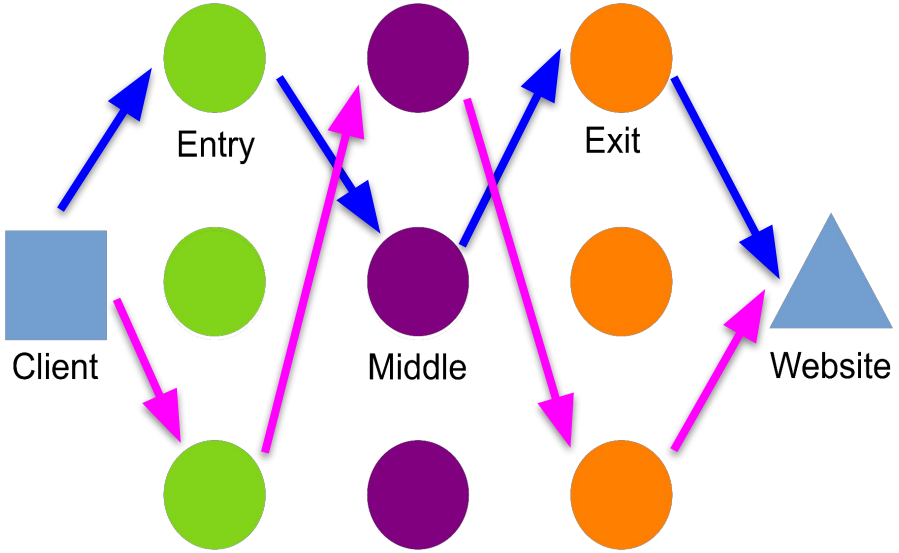
Tor: The Onion Router

- **Tor:** a distributed overlay network
  - Anonymizes TCP-based applications
    - Secure shell
    - Web browsing
    - Instant messaging
- Clients choose ???



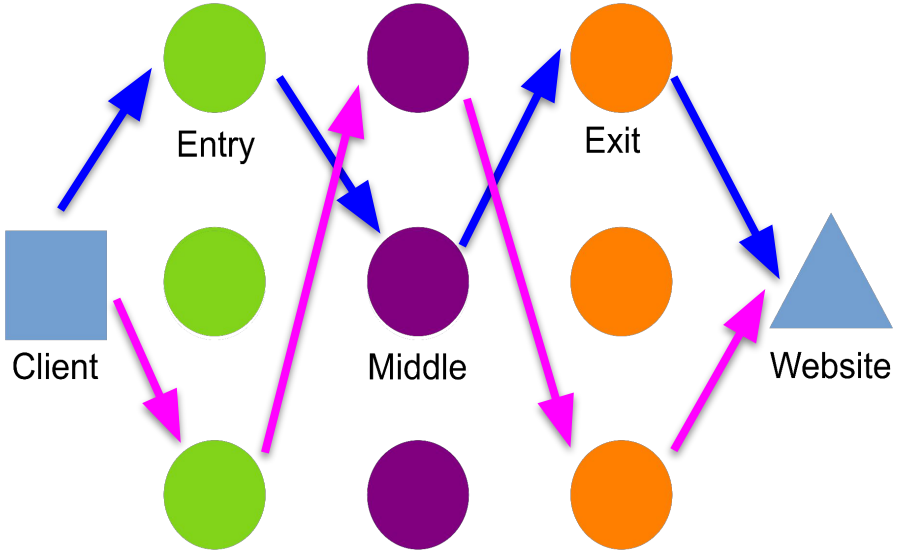
Tor: The Onion Router

- **Tor:** a distributed overlay network
  - Anonymizes TCP-based applications
    - Secure shell
    - Web browsing
    - Instant messaging
- Clients choose the **circuit paths**
  - Messages unwrapped at each onion router using a symmetric key
- Onion routers only know **???**

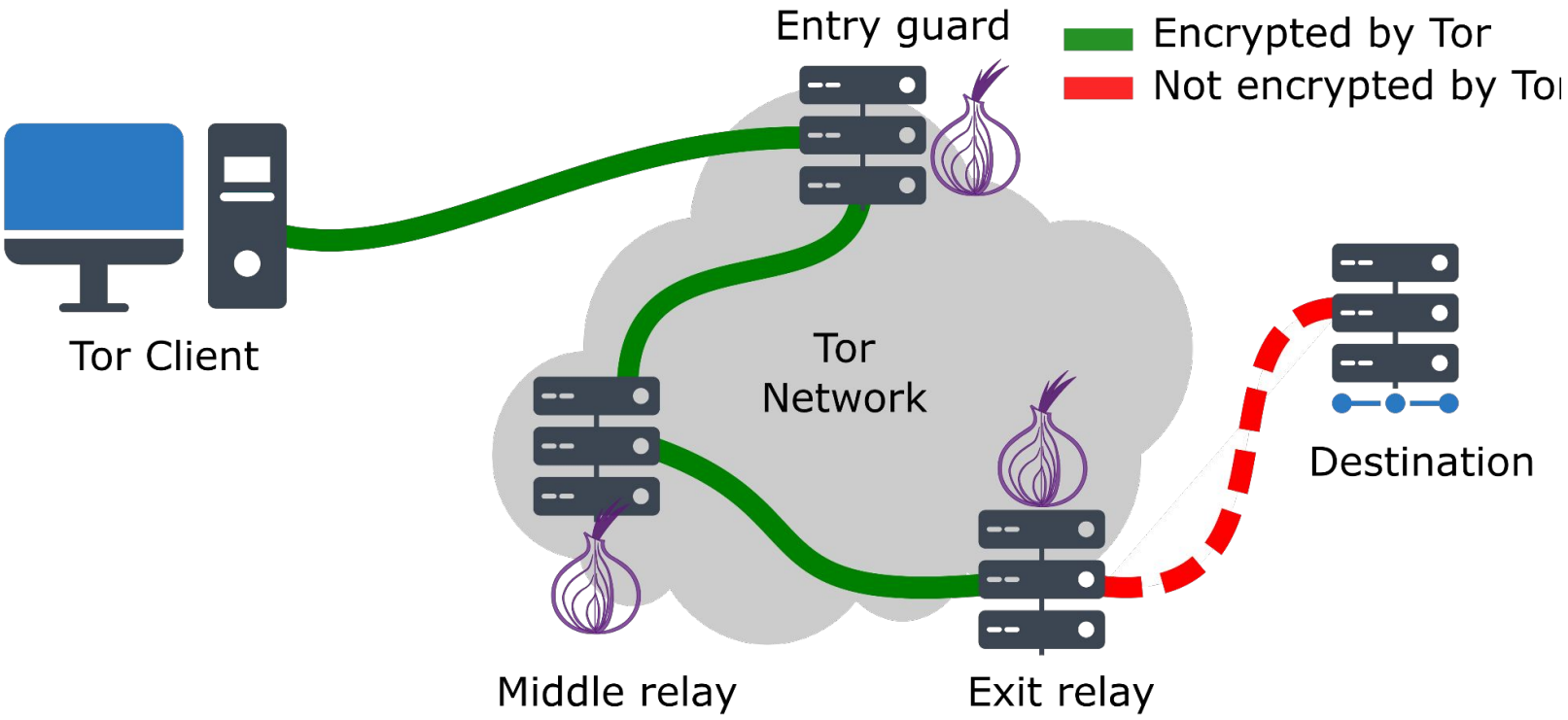


Tor: The Onion Router

- **Tor:** a distributed overlay network
  - Anonymizes TCP-based applications
    - Secure shell
    - Web browsing
    - Instant messaging
- Clients choose the **circuit paths**
  - Messages unwrapped at each onion router using a symmetric key
- Onion routers only know their **successor** or **predecessor** nodes
  - They don't know of any other nodes



How Tor Works



Attacking Tor

- **Possible attacks against Tor?**

Attacking Tor

- **Possible attacks against Tor?**
- **Leak DNS requests** when they aren't transmitted via Tor
- Perform **volume/timing analysis** to characterize behavior
- **Add malicious nodes** to intercept unencrypted exit traffic

Attacking Tor

- **Possible attacks against Tor?**
- **Leak DNS requests** when they aren't transmitted via Tor
  - **Defense: ???**
- Perform **volume/timing analysis** to characterize behavior
  - **Defense: ???**
- **Add malicious nodes** to intercept unencrypted exit traffic
  - **Defense: ???**

Attacking Tor

- **Possible attacks against Tor?**

- **Leak DNS requests** when they aren't transmitted via Tor
  - **Defense:** enforce all DNS requests **through Tor encryption**

- Perform **volume/timing analysis** to characterize behavior
  - **Defense:** inject **noisy data** to throw off analysis heuristics

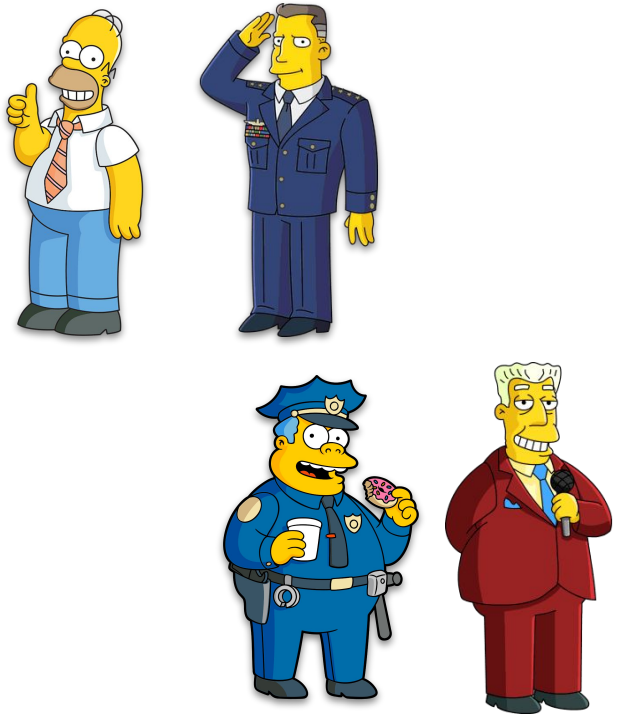
- **Add malicious nodes** to intercept unencrypted exit traffic
  - **Defense:** never use unencrypted protocols—**use HTTPS**

Who uses Tor?

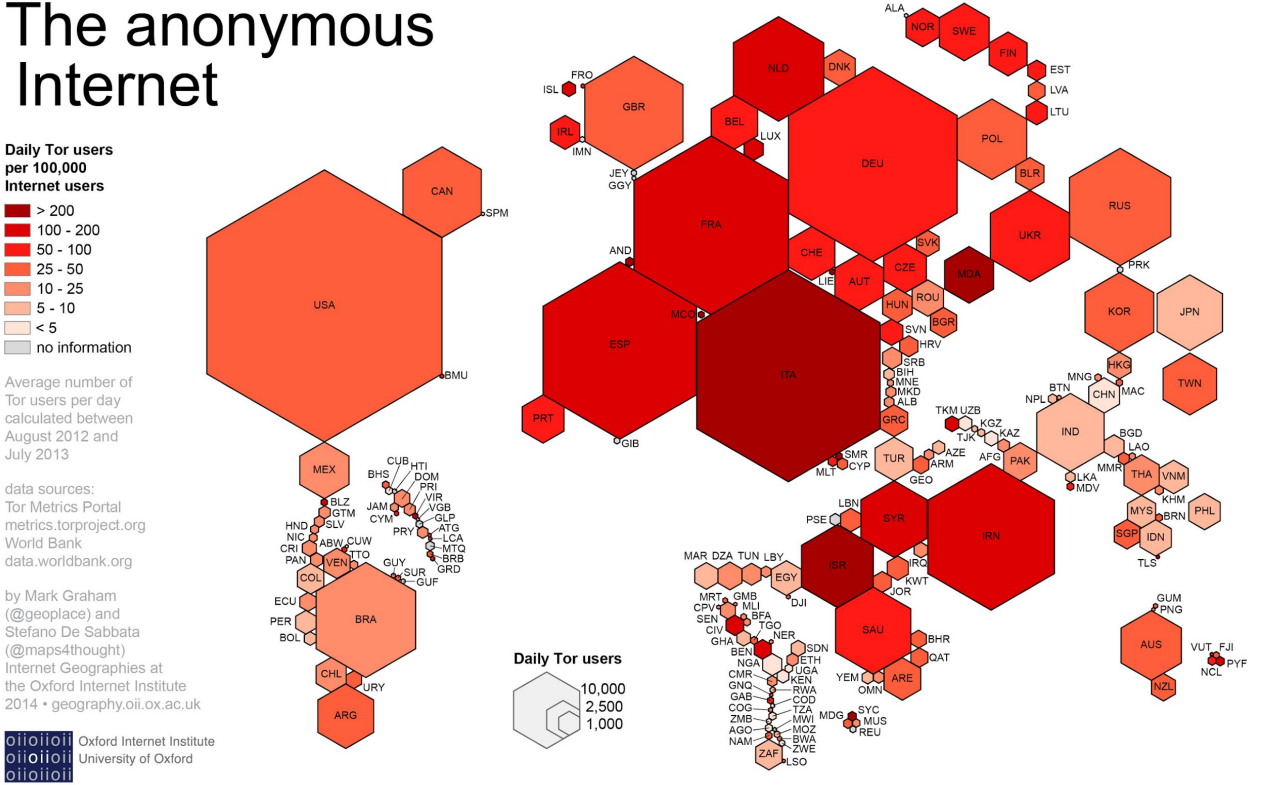
- ???

Who uses Tor?

- **Normal People**
  - Privacy-conscious folks
- **Intelligence Agencies**
  - Secret agents in the field
- **Law Enforcement**
  - Online “undercover” operations
- **Journalists and Bloggers**
  - Citizen journalists inspiring social change
- **Activists and Whistleblowers**
  - Raising their voice and avoiding persecution
- **White-hat and Black-hat Hackers**
  - And everyone in between!



Who uses Tor?


The anonymous Internet
Daily Tor users per 100,000 Internet users
> 200
100 - 200
50 - 100
25 - 50
10 - 25
5 - 10
< 5
no information
Average number of Tor users per day calculated between August 2012 and July 2013
data sources:
Tor Metrics Portal
metrics.torproject.org
World Bank
data.worldbank.org
by Mark Graham (@geoplace) and Stefano De Sabbata (@maps4thought)
Internet Geographies at the Oxford Internet Institute
2014 • geography.oii.ox.ac.uk
Oxford Internet Institute
University of Oxford
Daily Tor users
10,000
2,500
1,000

What services get hidden?

Positive Tor Use Cases

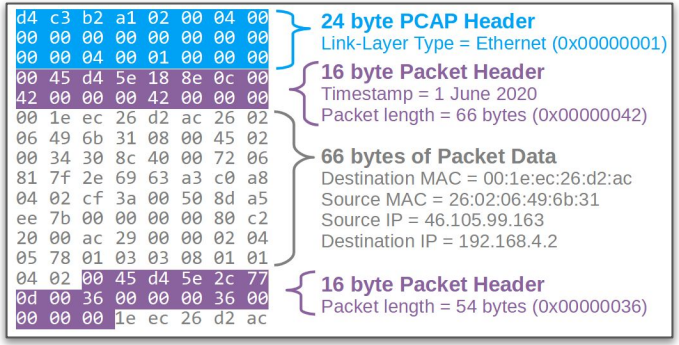
Questions?

Recap: Project 4 Overview

- Focuses on **network packet analysis**
  - Leveraging data contained within packets to achieve network defenses and attacks
- **Scenario:** helping a fictional university secure its enterprise campus network
  - Detect and characterizing likely attacks
  - Demonstrate how info can be intercepted

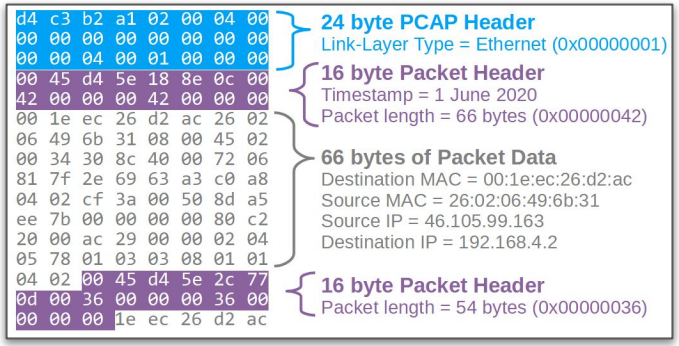
Recap: Project 4 Overview

- We provide a series of network packet traces (**pcaps**)
  - **Your job:** write scripts to analyze them!
- **Part 1:** detecting **network attacks**
  - Password cracking, port scanning, SYN floods
- **Part 2:** stealing **sensitive information**
  - Unencrypted credentials, browsing history
  - **Extra credit:** stealing transfered files



Recap: Project 4 Overview

- We provide a series of network packet traces (**pcaps**)
  - **Your job:** write scripts to analyze them!
- **Part 1:** detecting **network attacks**
  - Password cracking, port scanning, SYN floods
- **Part 2:** stealing **sensitive information**
  - Unencrypted credentials, browsing history
  - **Extra credit:** stealing transfered files
- You will use Python 3's **Scapy** library
  - A huge and powerful packet analysis API…
  - But we'll really only use **a few parts** of it



Recap: Scapy Fundamentals

- Python API for programmatic packet capture and analysis
  - Think of it as **"Wireshark in API form"**
- We provide **skeleton code** template
  - Sets-up the packet parsing workflow

```
#!/usr/bin/python3
import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *
import re

def parsePacket(packet):
    if not packet.haslayer("TCP"): return
    # ------------------------------------------
    # TODO: finish implementing parsePacket()!
    # ------------------------------------------
    return

if __name__ == "__main__":
    for packet in rdpcap(sys.argv[1]):
        parsePacket(packet)
```

Recap: Scapy Fundamentals

- Python API for programmatic packet capture and analysis
  - Think of it as **"Wireshark in API form"**
- We provide **skeleton code** template
  - Sets-up the packet parsing workflow
  - **Your job:** finish implementing the function `parsePacket()`

```
#!/usr/bin/python3
import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *
import re

def parsePacket(packet):
    if not packet.haslayer("TCP"): return
    # ------------------------------------------
    # TODO: finish implementing parsePacket()!
    # ------------------------------------------
    return

if __name__ == "__main__":
    for packet in rdpcap(sys.argv[1]):
        parsePacket(packet)
```

Recap: Scapy Fundamentals

- Python API for programmatic packet capture and analysis
  - Think of it as **"Wireshark in API form"**
- We provide **skeleton code** template
  - Sets-up the packet parsing workflow
  - **Your job:** finish implementing the function `parsePacket()`
- You may also add **additional code**
  - E.g., global variables or data structures
  - E.g., printing functionality in `main()`

```
#!/usr/bin/python3
import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *
import re

def parsePacket(packet):
    if not packet.haslayer("TCP"): return
    # -------------------------------------------
    # TODO: finish implementing parsePacket()!
    # -------------------------------------------
    return

if __name__ == "__main__":
    for packet in rdpcap(sys.argv[1]):
        parsePacket(packet)
```

Recap: Scapy Fundamentals

- Only a few things you'll need…
  - Get a packet's **TCP flags**:

```
packet["TCP"].flags
```

  - Get a packet's **destination port**

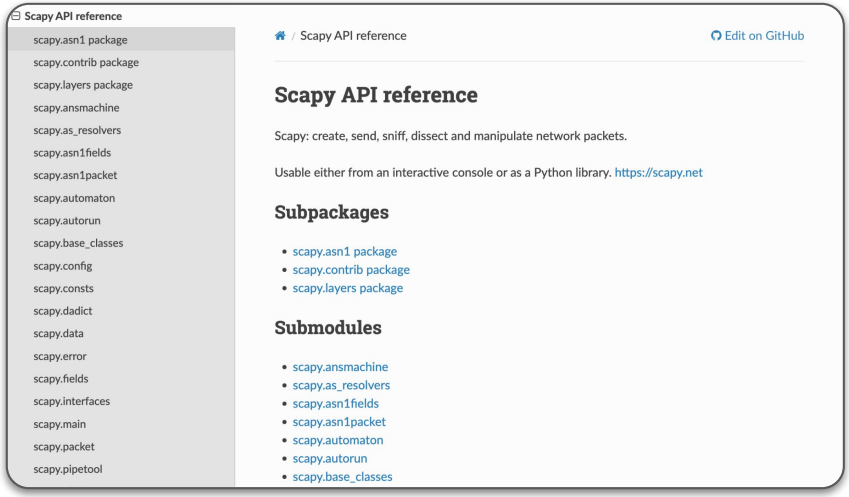
```
packet["TCP"].dport
```

  - Get a packet's **source IP address**

```
packet["IP"].src
```

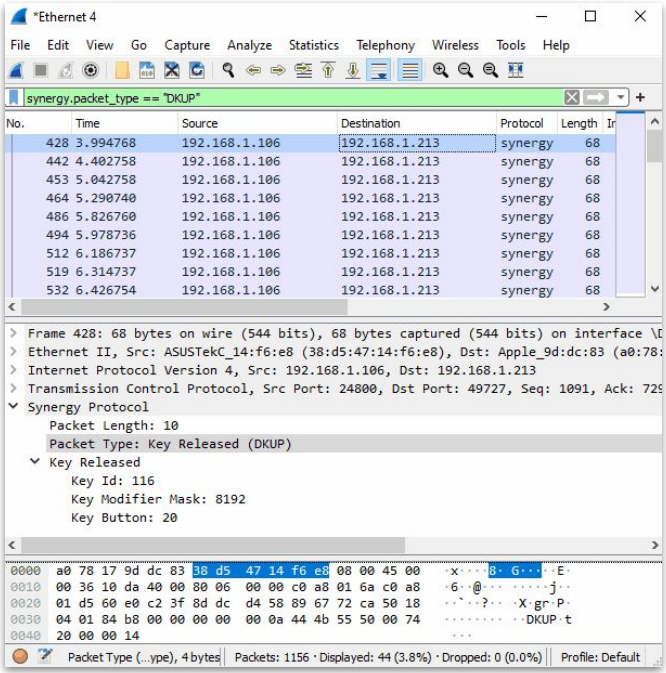
  - Get a packet's TCP **payload**:

```
bytes(packet["TCP"].payload).decode('utf-8','replace')
```



Recap: Suggested Workflow

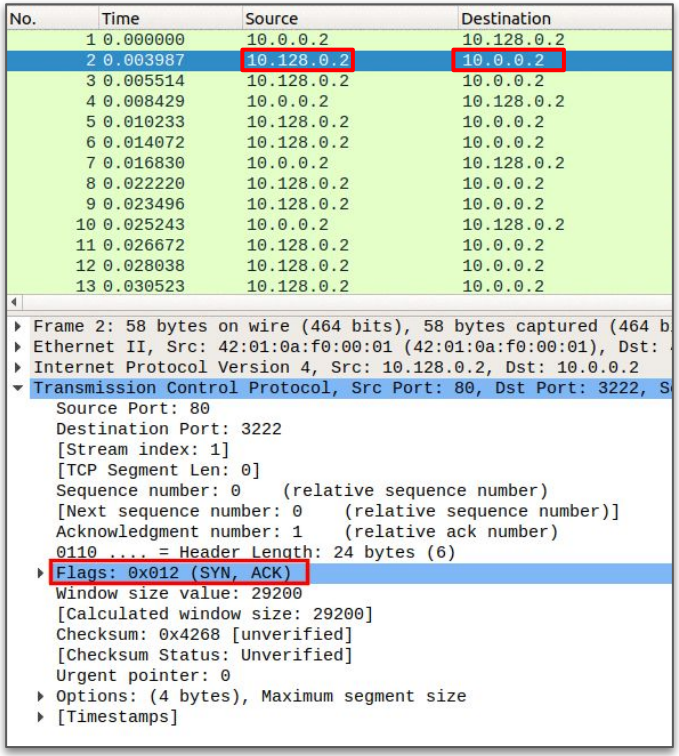
- Before you start writing a **Scapy** script, inspect the trace *manually* via **Wireshark**
  - Super helpful for viewing a packet's contents
  - Use this to bootstrap your script's approach!



Recap: Suggested Workflow

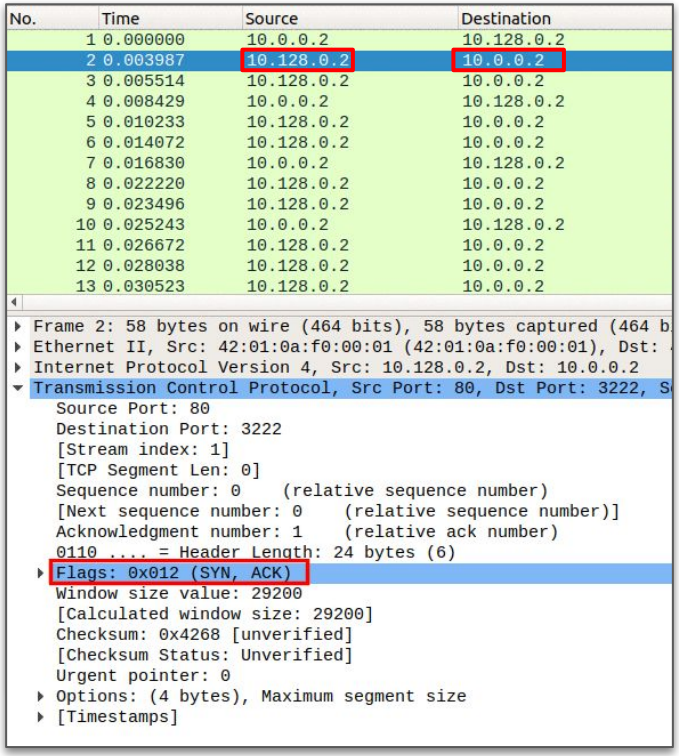
- Before you start writing a **Scapy** script, inspect the trace *manually* via **Wireshark**
  - Super helpful for viewing a packet's contents
  - Use this to bootstrap your script's approach!

- For each target, answer the following:
  - What **packet fields** matter?
  - How to **extract** relevant data?
  - How to **store and process** this data?



Recap: Suggested Workflow

- Before you start writing a **Scapy** script, inspect the trace *manually* via **Wireshark**
  - Super helpful for viewing a packet's contents
  - Use this to bootstrap your script's approach!
- For each target, answer the following:
  - What **packet fields** matter?
  - How to **extract** relevant data?
  - How to **store and process** this data?
- Finalize your **high-level game plan** first!
  - Then start developing your solution scripts!



Questions?

This time on CS 4440...

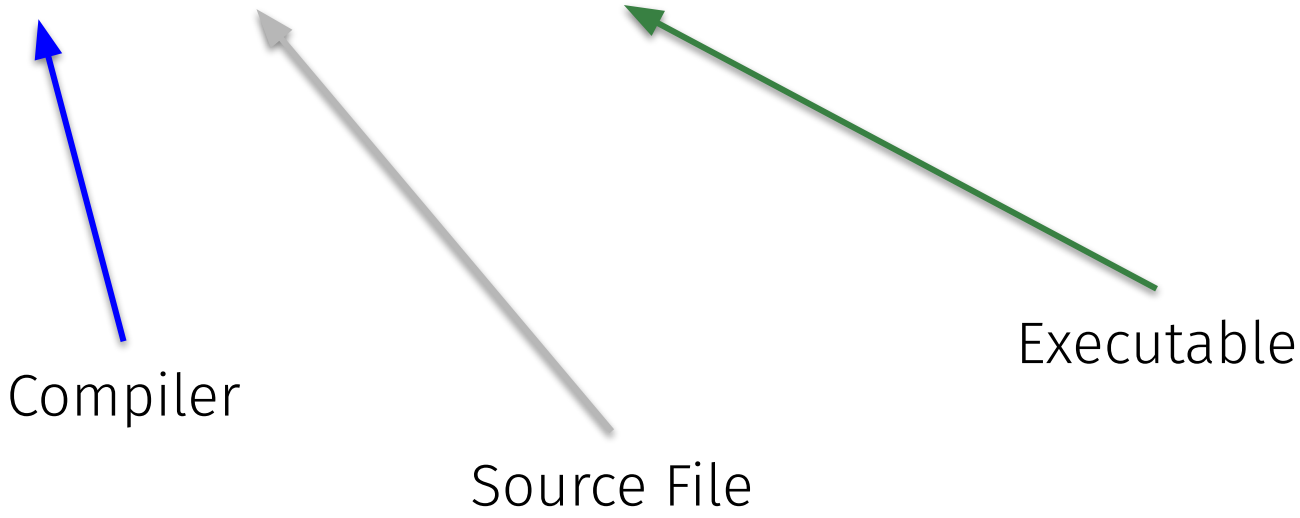
Binary Reverse Engineering
Instruction Recovery
Control Flow Analysis
Structure Recovery
RE Challenges

How Software is Built

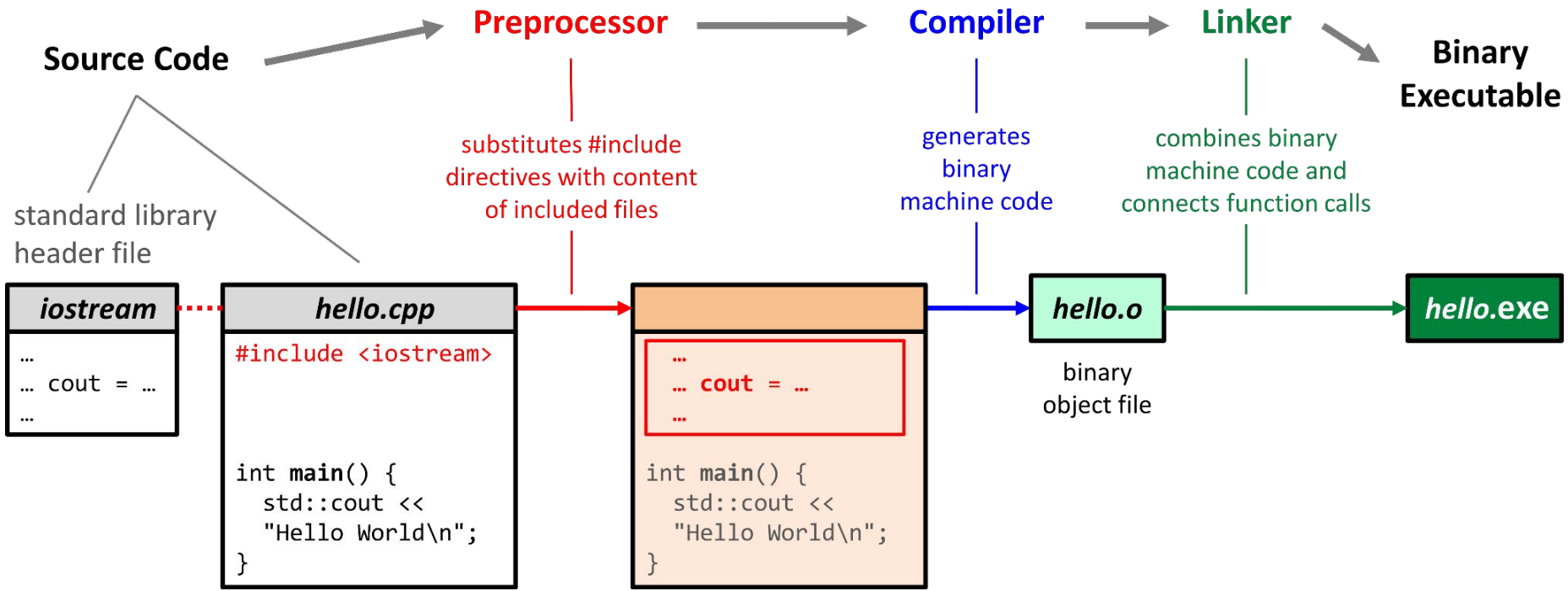
- `clang hello.c -o hello`

How Software is Built

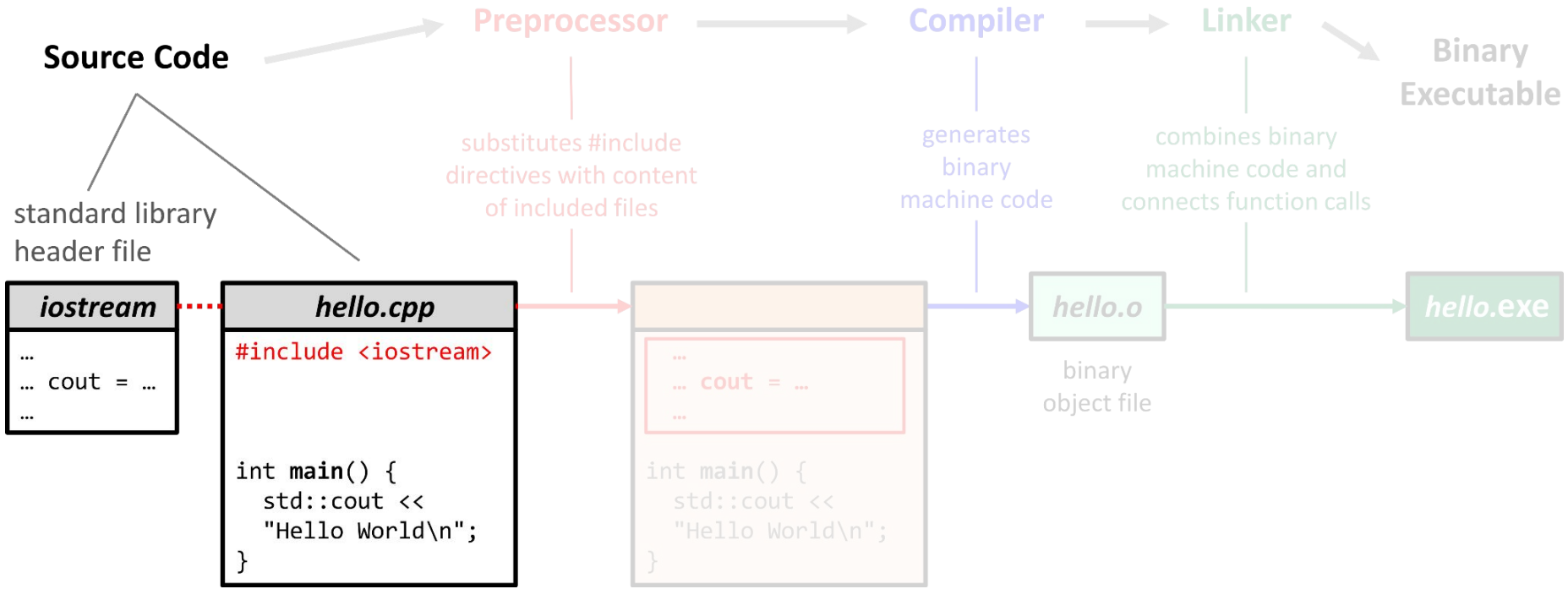
- `clang hello.c -o hello`



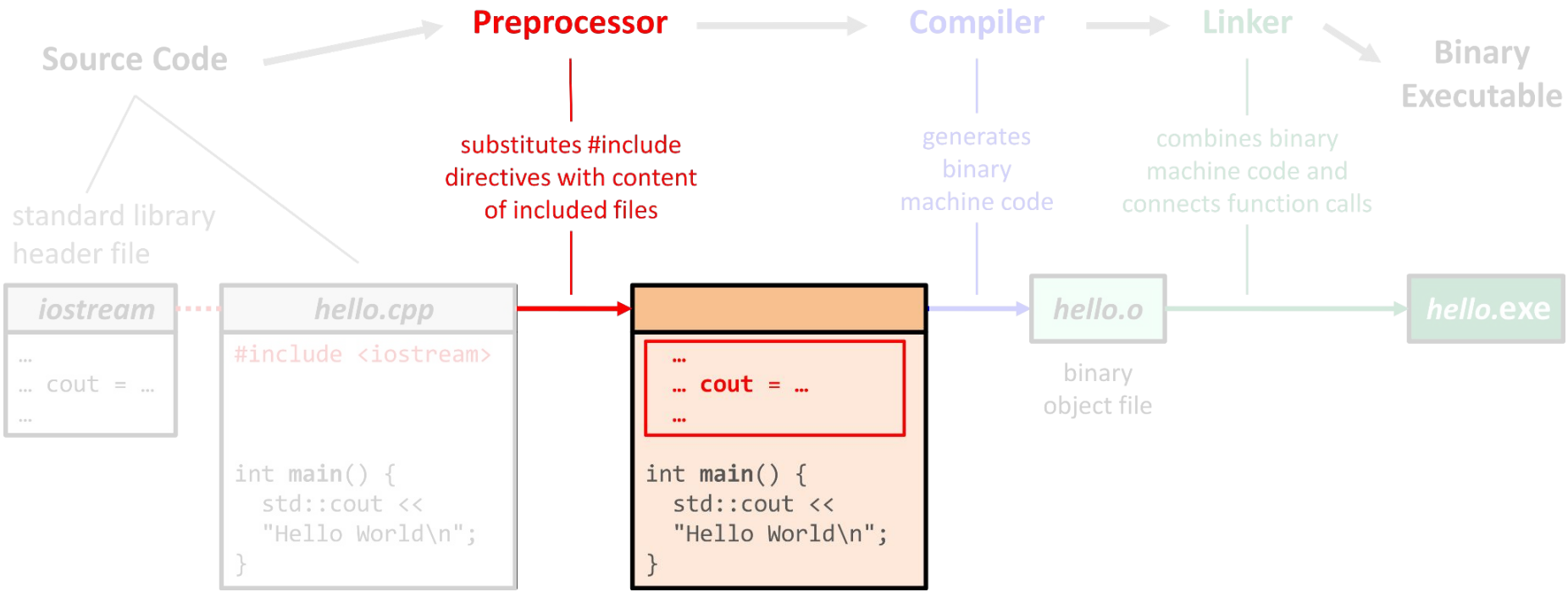
How Software is Built



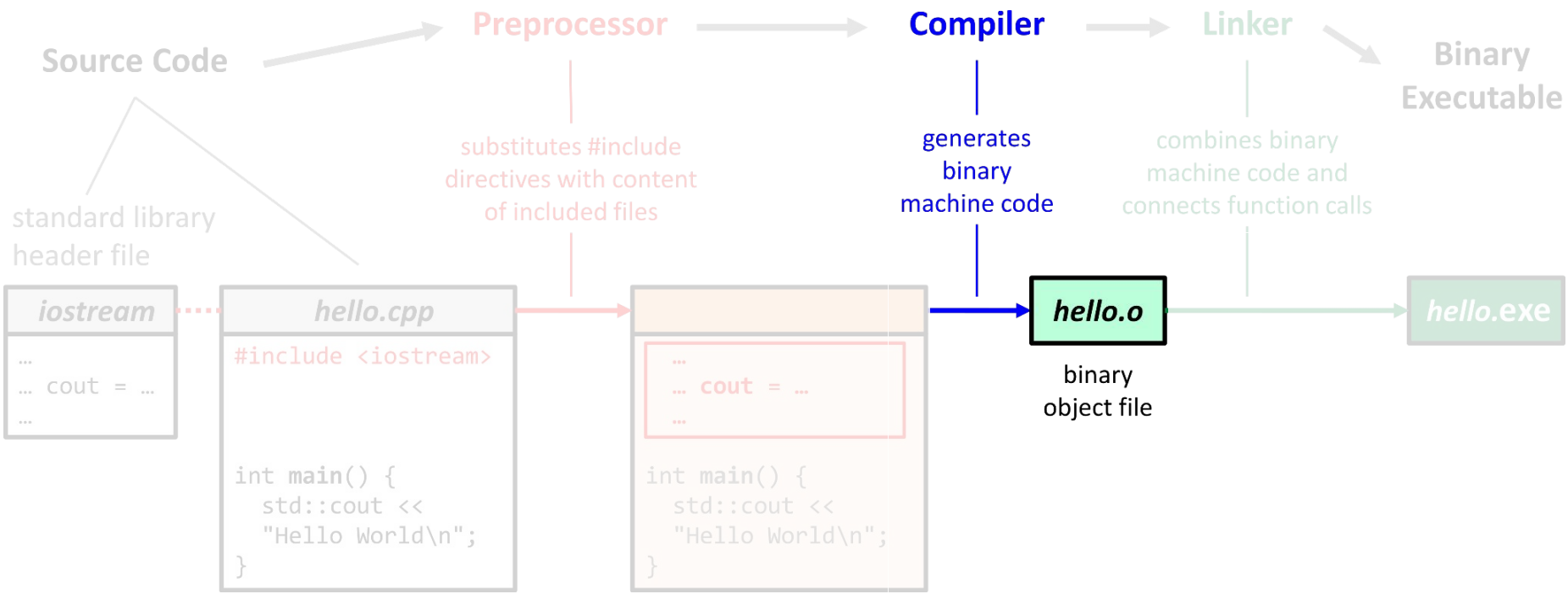
How Software is Built



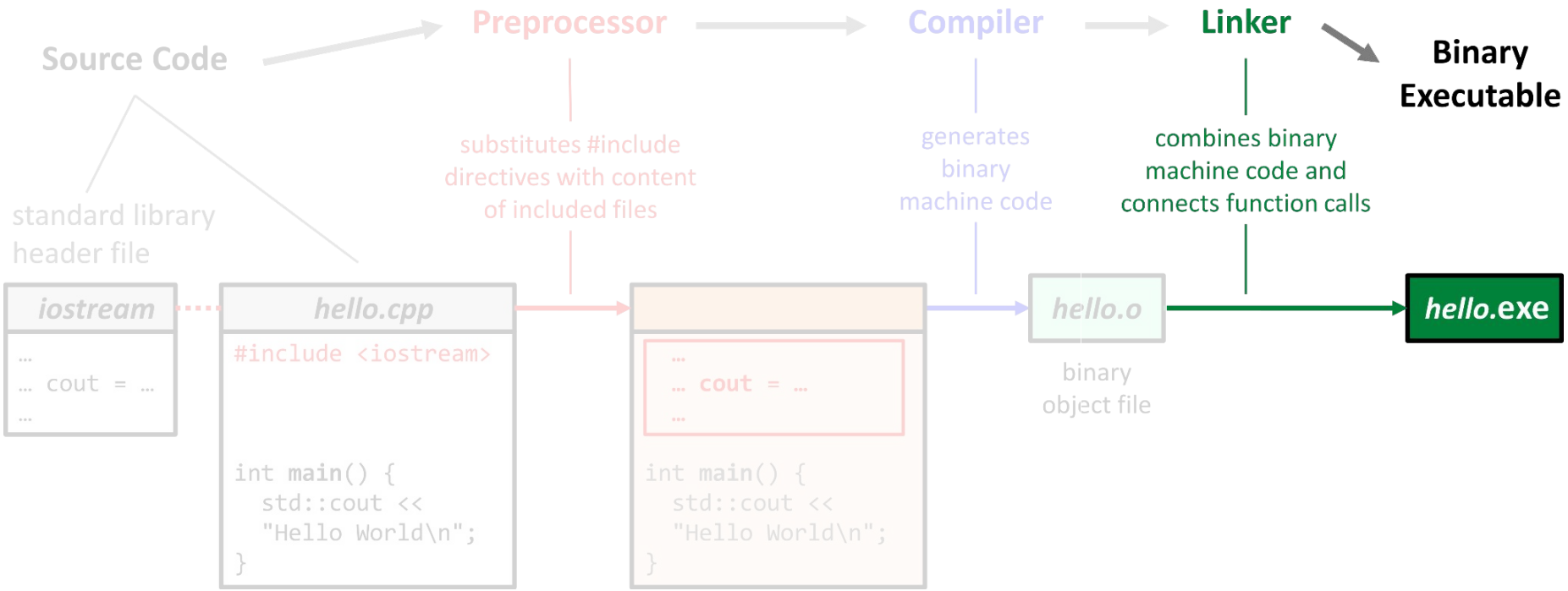
How Software is Built



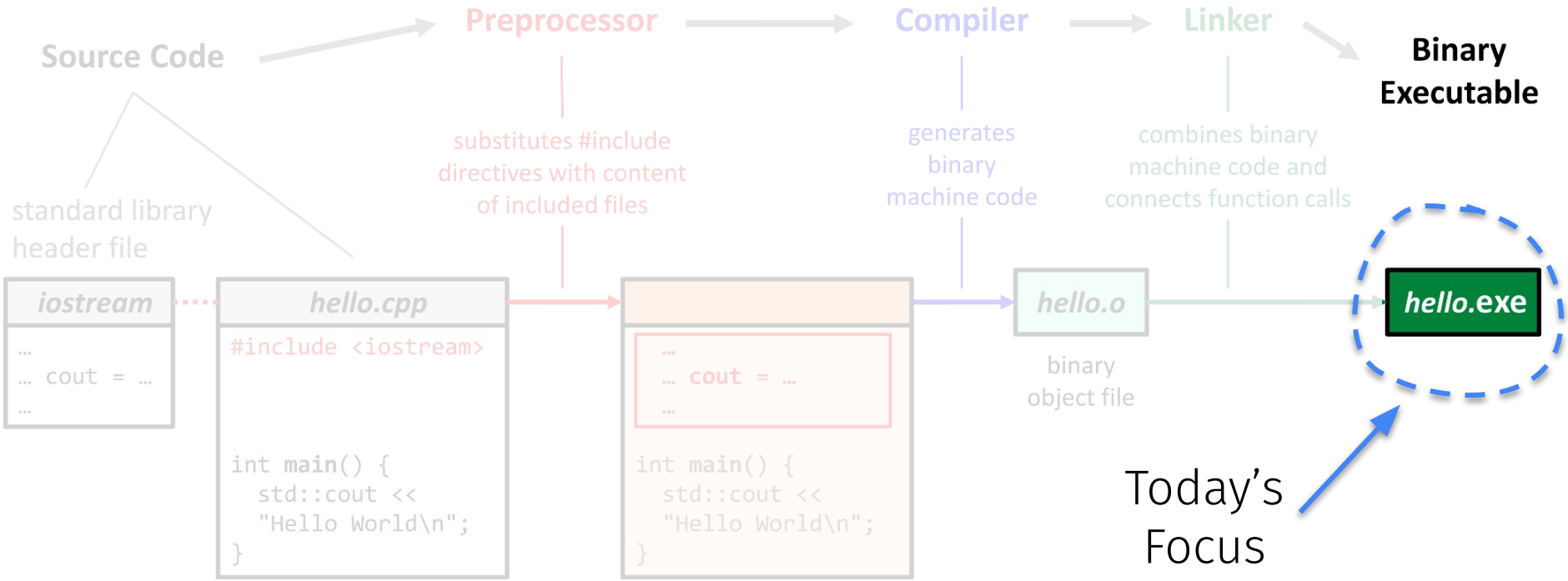
How Software is Built



How Software is Built



How Software is Built



Examples of Closed-source Software?

Nobody has responded yet.

Hang tight! Responses are coming in.

Closed-source Software

- **It's everywhere!**



Closed-source Software

- It's everywhere!

Commercialized applications and libraries

Freely-distributed **proprietary software**

Legacy software whose source code is lost

Auditing Open- versus Closed-source Code

Open Source:

- **Publicly-available** source codebase
- Achieves security by **transparency**

- **Semantic richness** facilitates **high-performance**, **effective** vetting

Auditing Open- versus Closed-source Code

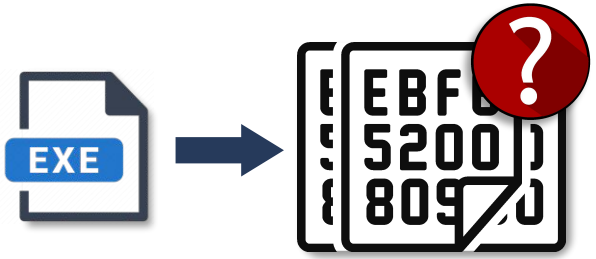
Open Source:

- **Publicly-available** source codebase
- Achieves security by **transparency**

- **Semantic richness** facilitates **high-performance**, **effective** vetting

Closed Source:

- Distributed as a precompiled **binary**
- **Opaque to *everyone*** but its developer



- **Upwards of 10x slower** security vetting
- Forced to **rely on crude techniques**

Auditing Open- versus Closed-source Code

- Global market size **over $240 billion**
- **85%** contains **critical** vulnerabilities
- **89%** of the **most exploited** software

Closed Source:

- Distributed as a precompiled **binary**
- Opaque to *everyone* but its developer

- Upwards of **10x slower** security vetting
- Forced to **rely on crude techniques**

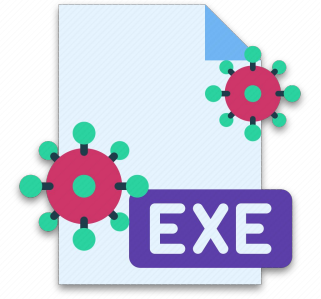
Reverse Engineering (RE)

- **What is RE?**

> "A process or method through which one attempts to **understand** through deductive reasoning how a previously made **device**, **process**, **system**, or piece of **software** accomplishes a task with **very little (if any) insight** into exactly how it does so."

Why do we care about RE?

- **Discovering bugs**
- **Retrofitting fixes**
- **Malware analysis**
- **Right to repair!**



RE Tasks

- **Disassembly**
  - ???

RE Tasks

- **Disassembly**
  - Machine code to human readable assembly
- **Decompilation**
  - ???

```
00000003  4F                dec edi
00000004  6A1E              push byte +0x1e
00000006  B7B5              mov bh,0xb5
00000008  0C12              or al,0x12
0000000A  6A04              push byte +0x4
0000000C  EAA08EA57B2BB1    jmp dword 0xb12b:0x7ba58ea0
00000013  B114              mov cl,0x14
```

RE Tasks

- **Disassembly**
  - Machine code to human readable assembly
- **Decompilation**
  - Machine code to human readable source code
- **Rewriting**
  - ???

```
00000003  4F                dec edi
00000004  6A1E              push byte +0x1e
00000006  B7B5              mov bh,0xb5
00000008  0C12              or al,0x12
0000000A  6A04              push byte +0x4
0000000C  EAA08EA57B2BB1    jmp dword 0xb12b:0x7ba58ea0
00000013  B114              mov cl,0x14
```

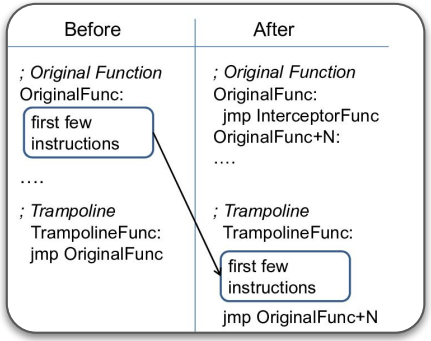
```
2  int main(void)
3
4  {
5    char local_54 [64];
6    int modified;
7
8    modified = 0;
9    gets(local_54);
10   if (modified == 0) {
11     puts("Try again?");
12   }
13   else {
14     puts("you have changed the \'modified\' variable");
15   }
16   return 0;
17 }
18
```

RE Tasks

- **Disassembly**
  - Machine code to human readable assembly
- **Decompilation**
  - Machine code to human readable source code
- **Rewriting**
  - Add more functionality and rebuild executable

```
00000003  4F                dec edi
00000004  6A1E              push byte +0x1e
00000006  B7B5              mov bh,0xb5
00000008  0C12              or al,0x12
0000000A  6A04              push byte +0x4
0000000C  EAA08EA57B2BB1    jmp dword 0xb12b:0x7ba58ea0
00000013  B114              mov cl,0x14
```

```
2  int main(void)
3
4  {
5    char local_54 [64];
6    int modified;
7
8    modified = 0;
9    gets(local_54);
10   if (modified == 0) {
11     puts("Try again?");
12   }
13   else {
14     puts("you have changed the \'modified\' variable");
15   }
16   return 0;
17 }
18
```



Experience with RE?

Disassembly (e.g., objdump)

0%

Decompilation (e.g., Ghidra, IDA)

0%

Something else!

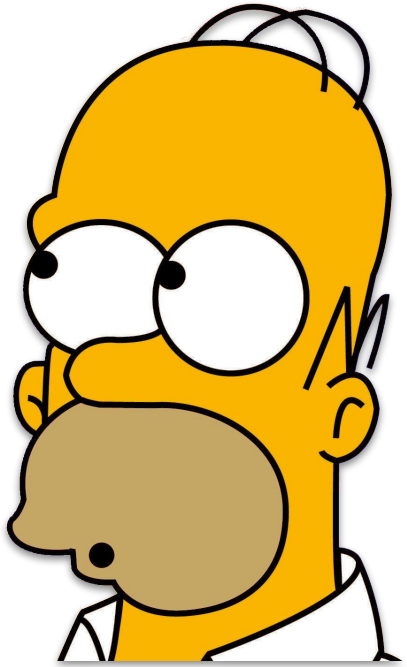
0%

None of the above (totally fine!)

0%

Three Pillars of RE

1. **Instruction Recovery**
   - Decode bytes to instructions
   - Disambiguate code from data
2. **Control Flow Recovery**
   - Intra-procedural execution flow
   - Inter-procedural execution flow
3. **Program Structure Recovery**
   - Identify program basic blocks
   - Higher-level constructs (e.g., loops)



Questions?

Pillars of RE: Instruction Recovery

Instructions

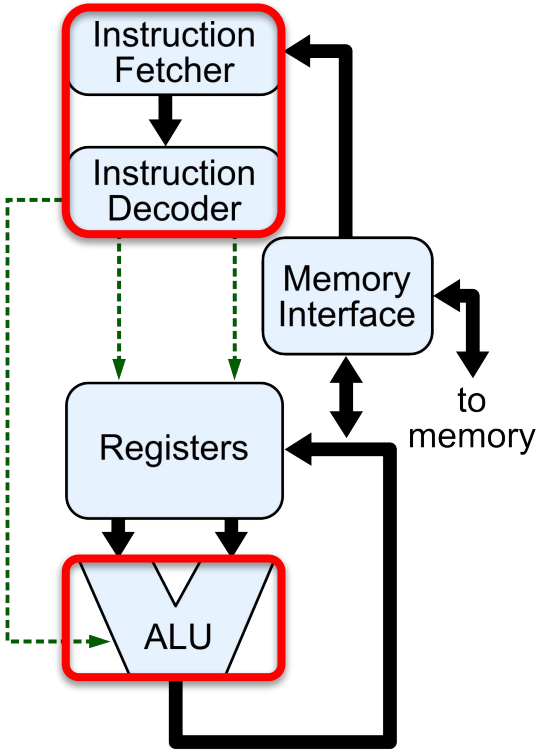
- **What are they?**

MAXPS	Maximum of Packed Single-Precision Floating-Point Values
MAXSD	Return Maximum Scalar Double-Precision Floating-Point Value
MAXSS	Return Maximum Scalar Single-Precision Floating-Point Value
MFENCE	Memory Fence
MINPD	Minimum of Packed Double-Precision Floating-Point Values
MINPS	Minimum of Packed Single-Precision Floating-Point Values
MINSD	Return Minimum Scalar Double-Precision Floating-Point Value
MINSS	Return Minimum Scalar Single-Precision Floating-Point Value
MONITOR	Set Up Monitor Address
MOV	Move
MOV (1)	Move to/from Control Registers
MOV (2)	Move to/from Debug Registers
MOVAPD	Move Aligned Packed Double-Precision Floating-Point Values
MOVAPS	Move Aligned Packed Single-Precision Floating-Point Values
MOVBE	Move Data After Swapping Bytes
MOVD	Move Doubleword/Move Quadword
MOVDDUP	Replicate Double FP Values
MOVDIR64B	Move 64 Bytes as Direct Store
MOVDIRI	Move Doubleword as Direct Store
MOVDQ2Q	Move Quadword from XMM to MMX Technology Register

Recap: The CPU

- State modified by assembly **instructions**
  - ADD, SUB, XOR, CMP, CALL, JMP, RET
  - **And many more!**

- Assembly instruction syntaxes
  - **AT&T** = `Instruction Source Destination`
  - **Intel** = `Instruction Destination Source`
  - Example: **MOV** SRC, DST versus **MOV** DST, SRC
  - This lecture: **AT&T syntax**



Instructions

- **What are they?**
  - Operations that modify **CPU state**
- Source = **???**
- x86 asm = **???**

MAXPS	Maximum of Packed Single-Precision Floating-Point Values
MAXSD	Return Maximum Scalar Double-Precision Floating-Point Value
MAXSS	Return Maximum Scalar Single-Precision Floating-Point Value
MFENCE	Memory Fence
MINPD	Minimum of Packed Double-Precision Floating-Point Values
MINPS	Minimum of Packed Single-Precision Floating-Point Values
MINSD	Return Minimum Scalar Double-Precision Floating-Point Value
MINSS	Return Minimum Scalar Single-Precision Floating-Point Value
MONITOR	Set Up Monitor Address
MOV	Move
MOV (1)	Move to/from Control Registers
MOV (2)	Move to/from Debug Registers
MOVAPD	Move Aligned Packed Double-Precision Floating-Point Values
MOVAPS	Move Aligned Packed Single-Precision Floating-Point Values
MOVBE	Move Data After Swapping Bytes
MOVD	Move Doubleword/Move Quadword
MOVDDUP	Replicate Double FP Values
MOVDIR64B	Move 64 Bytes as Direct Store
MOVDIRI	Move Doubleword as Direct Store
MOVDQ2Q	Move Quadword from XMM to MMX Technology Register

Instructions

- **What are they?**
  - Operations that modify **CPU state**
- Source = **high-level** instructions
  - Human-readable
- x86 asm = **low-level** instructions
  - Somewhat human-readable

Key to inferring what the program is doing

MAXPS	Maximum of Packed Single-Precision Floating-Point Values
MAXSD	Return Maximum Scalar Double-Precision Floating-Point Value
MAXSS	Return Maximum Scalar Single-Precision Floating-Point Value
MFENCE	Memory Fence
MINPD	Minimum of Packed Double-Precision Floating-Point Values
MINPS	Minimum of Packed Single-Precision Floating-Point Values
MINSD	Return Minimum Scalar Double-Precision Floating-Point Value
MINSS	Return Minimum Scalar Single-Precision Floating-Point Value
MONITOR	Set Up Monitor Address
MOV	Move
MOV (1)	Move to/from Control Registers
MOV (2)	Move to/from Debug Registers
MOVAPD	Move Aligned Packed Double-Precision Floating-Point Values
MOVAPS	Move Aligned Packed Single-Precision Floating-Point Values
MOVBE	Move Data After Swapping Bytes
MOVD	Move Doubleword/Move Quadword
MOVDDUP	Replicate Double FP Values
MOVDIR64B	Move 64 Bytes as Direct Store
MOVDIRI	Move Doubleword as Direct Store
MOVDQ2Q	Move Quadword from XMM to MMX Technology Register

Recovering Instructions

- **Goal:** translate bytes into **logical instructions**
  - Called instruction **decoding**
  - Analogous to what CPU does
  - General output: **disassembly**

Instruction stream

```
B8 22 11 00 FF 01 CA 31 F6 53 8B 5C 24
04 8D 34 48 39 C3 72 EB C3
```

Read bytes from input executable

Machine code bytes

```
B8 22 11 00 FF
01 CA
31 F6
53
8B 5C 24 04
8D 34 48
39 C3
72 EB
C3
```

Group bytes

Assembly language statements

```
foo:
movl $0xFF001122, %eax
addl %ecx, %edx
xorl %esi, %esi
pushl %ebx
movl 4(%esp), %ebx
leal (%eax,%ecx,2), %esi
cmpl %eax, %ebx
jnae foo
retl
```

Decode instructions

Instruction Recovery Techniques

- **Linear Sweep**
  - Start decoding at binary entry
  - Attempt to decode all bytes
  - Stop at end of .TEXT section

Intuition: compilers lay code sequentially for compactness

Challenge: data within code

Instruction Recovery Techniques

- **Linear Sweep**
  - Start decoding at binary entry
  - Attempt to decode all bytes
  - Stop at end of .TEXT section

Intuition: compilers lay code sequentially for compactness

Challenge: data within code

- **Recursive Descent**
  - Follow all control-flow transfers
  - `jmp 0x100` → start decoding instructions at address `0x100`
  - Stop when you've covered all possible control-flow paths

Intuition: following the logical flow of execution reveals a lot

Challenge: indirect branches

Instruction Recovery Techniques

- **Linear Sweep**
  - Start decoding at binary entry
  - Attempt to decode all bytes
  - Stop at end of .TEXT section

Intuition: compilers lay code sequentially for compactness

- **Recursive D**
  - Follow all
  - `jmp 0x100` → start decoding instructions at address `0x100`
  - Stop when you've covered all possible control-flow paths

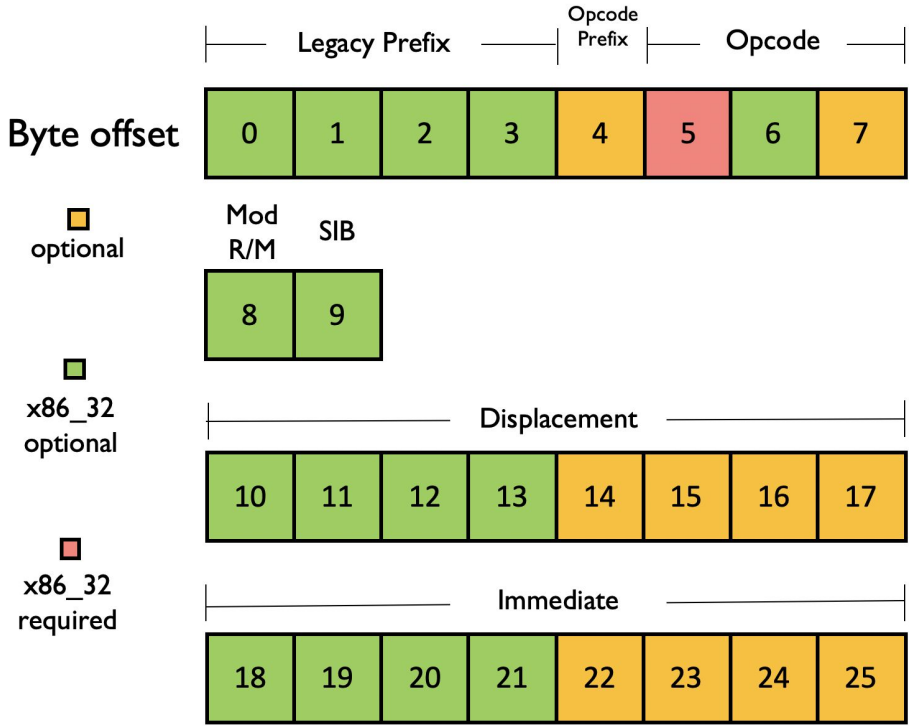
Intuition: following the logical flow of execution reveals a lot

Challenge: indirect branches

Most modern RE adopts a **combined** approach in addition to **heuristics**

CISC Architectures

- **Variable-length** instructions
  - E.g., x86-32, x86-64
- Almost any byte sequence can be a **valid instruction**!
- Being just one byte off can totally mess up decoding!



CISC Architectures

- Example of byte offsets and possible decodings:

```
0x0F 0x88 0x52 0x0F 0x84 0xEC
```

```
js      0xffffffffec840f58
```

CISC Architectures

- Example of byte offsets and possible decodings:

```
0x0F 0x88 0x52 0x0F 0x84 0xEC
```

```
mov     BYTE PTR [rdx+0xf],dl
test    ah,ch
```

CISC Architectures

- Example of byte offsets and possible decodings:

```
0x0F 0x88 0x52 0x0F 0x84 0xEC
```

```
add     eax,0x40080f20
in      al,dx
```

Instruction Decoder Bugs

- Results from Trail of Bits' Mishegos fuzzer:

input	worker			
	./src/worker/bfd/bfd.so	./src/worker/capstone/capstone.so	./src/worker/xed/xed.so	./src/worker/zydis/zydis.so
6567f2f39cb2a58654	gs addr32 repnz repz pushf (5 / 28)	pushfq (5 / 8)	addr32 pushfq (5 / 14)	pushfq (5 / 6)
26f267664d0f3817314aecdf9d	es addr32 data16 rex.WRB (bad) (8 / 32)	ptest xmm14, xmmword ptr es:[r9d] (9 / 34)	(0 / 0)	(0 / 0)
656636f26f0f3a6f959066b1fd8c52	gs repnz outs dx,WORD PTR ss:[rsi] (5 / 35)	repne outsd dx, dword ptr ss:[rsi] (5 / 35)	repne data16 outsw gs (5 / 21)	repne outsw (5 / 11)
67f03ef0460f1104fe	lock lock movups XMMWORD PTR ds:[esi+r15d*8],xmm8 (9 / 50)	(0 / 0)	(0 / 0)	(0 / 0)
652e26520ffda71fd5bc9e3090235f	gs cs es push rdx (4 / 18)	push rdx (4 / 9)	push rdx (4 / 8)	push rdx (4 / 8)
64652e26520ffda71fd5bc9e309023	fs gs cs es push rdx (5 / 21)	push rdx (5 / 9)	push rdx (5 / 8)	push rdx (5 / 8)
6636f26f0f3a6f959066b1fd8c523e	repnz outs dx,WORD PTR ss:[rsi] (4 / 32)	repne outsd dx, dword ptr ss:[rsi] (4 / 35)	repne data16 outsw (4 / 19)	repne outsw (4 / 11)
2e26520ffda71fd5bc9e3090235f0d	cs es push rdx (3 / 15)	push rdx (3 / 9)	push rdx (3 / 8)	push rdx (3 / 8)
36f2f3f00f3a6315cd	ss repnz lock (bad) (7 / 21)	(0 / 0)	(0 / 0)	(0 / 0)
262ef0f00f3ada4eb5bae8	es cs lock lock (bad) (7 / 23)	(0 / 0)	(0 / 0)	(0 / 0)
f365f33e9559dd95a732b088057275	repz gs repz ds xchg ebp,eax (5 / 29)	xchg eax, ebp (5 / 14)	xchg ebp, eax (5 / 13)	xchg ebp, eax (5 / 13)
65f33e9559dd95a732b08805727509	gs repz ds xchg ebp,eax (4 / 24)	xchg eax, ebp (4 / 14)	xchg ebp, eax (4 / 13)	xchg ebp, eax (4 / 13)
f33e9559dd95a732b08805727509f3	repz ds xchg ebp,eax (3 / 21)	xchg eax, ebp (3 / 14)	xchg ebp, eax (3 / 13)	xchg ebp, eax (3 / 13)
3e9559dd95a732b08805727509f395	ds xchg ebp,eax (2 / 16)	xchg eax, ebp (2 / 14)	xchg ebp, eax (2 / 13)	xchg ebp, eax (2 / 13)
6765676547be4b69	addr32 (1 / 7)	(0 / 0)	(0 / 0)	(0 / 0)
59dd95a732b08805727509f39507d0	pop rcx (1 / 11)	pop rcx (1 / 8)	pop rcx (1 / 7)	pop rcx (1 / 7)
3e3e2e265ddf3b	ds ds cs es pop rbp (5 / 20)	pop rbp (5 / 8)	pop rbp (5 / 7)	pop rbp (5 / 7)
9559dd95a732b08805727509f39507	xchg ebp,eax (1 / 15)	xchg eax, ebp (1 / 14)	xchg ebp, eax (1 / 13)	xchg ebp, eax (1 / 13)
646526f04991a85e	fs gs es lock xchg r9,rax (6 / 26)	(0 / 0)	(0 / 0)	(0 / 0)
f26426644a0f38d5fbbe	repnz fs es fs rex.WX (bad) (8 / 29)	(0 / 0)	(0 / 0)	(0 / 0)
64666566950f3a13f9f6	fs data16 gs xchg bp,ax (5 / 24)	xchg ax, bp (5 / 12)	xchg bp, ax (5 / 11)	xchg bp, ax (5 / 11)
f02ef065440f5a52e209f49611d758	lock cs lock cvtps2pd xmm10,QWORD PTR gs:[rdx-0x1e] (9 / 52)	(0 / 0)	(0 / 0)	(0 / 0)
3626646747 0f3875ae022de4a1517e	es es fs addr32 rex.RXB (bad) (8 / 31)	(0 / 0)	(0 / 0)	(0 / 0)
266467470f3875ae022de4a1517e90	es fs addr32 rex.RXB (bad) (7 / 28)	(0 / 0)	(0 / 0)	(0 / 0)
6467470f3875ae022de4a1517e90b4	fs addr32 rex.RXB (bad) (6 / 25)	(0 / 0)	(0 / 0)	(0 / 0)
f3f32666cd0f38b0c9a1ef83f720	repz repz es data16 int 0xf (6 / 28)	int 0xf (6 / 8)	int 0xf (6 / 7)	int 0x0F (6 / 8)
2ef066f0480f3a2904b7	es lock data16 lock rex.W (bad) (8 / 33)	(0 / 0)	(0 / 0)	(0 / 0)
67470f3875ae022de4a1517e90b4cb	addr32 rex.RXB (bad) (5 / 22)	(0 / 0)	(0 / 0)	(0 / 0)
3664970f7a50db978a650a8288bee1	es fs xchg edi,eax (3 / 19)	xchg eax, edi (3 / 14)	xchg edi, eax (3 / 13)	xchg edi, eax (3 / 13)
f226f236458a49fb848d28	repnz es repnz mov r9b,BYTE PTR ss:[r9-0x5] (8 / 44)	mov r9b, byte ptr ss:[r9 - 5] (8 / 30)	mov r9b, byte ptr [r9-0x5] (8 / 26)	mov r9b, ss:[r9-0x05] (8 / 21)

Code vs. Data

- **Some compilers** tightly interweave **data** (e.g., bytes, values) **within code**
  - Imprecision can create **trickle-down errors** in instruction recovery!
  - Example from OpenSSL (one of the most popular HTTPS libraries):

```
popfq           // original
.byte 0xf3,0xc3
.size AES_cbc_encrypt
.align 64
.LAES_Te
.long 0xa56363c6
```

```
popfq     // disassembled
repz retq
nop
nop
(bad)
movslq -0x5b(%rbx),%esp
```

Questions?

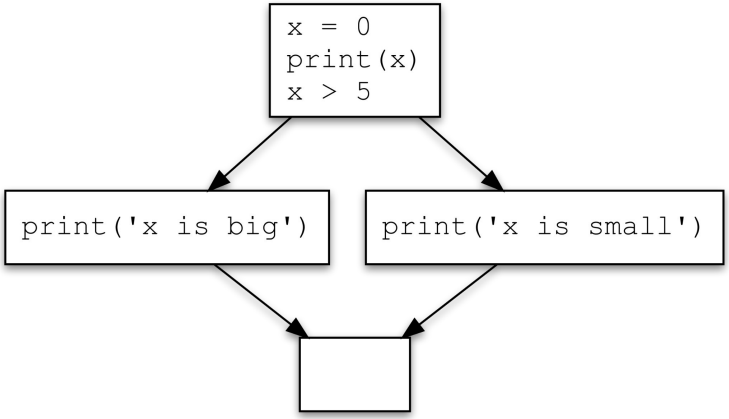
Pillars of RE:
Control Flow Recovery

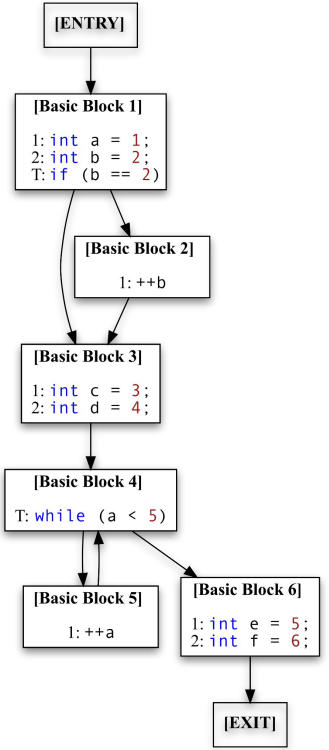
Control Flow

- **What is it?**
  - ???

Control Flow

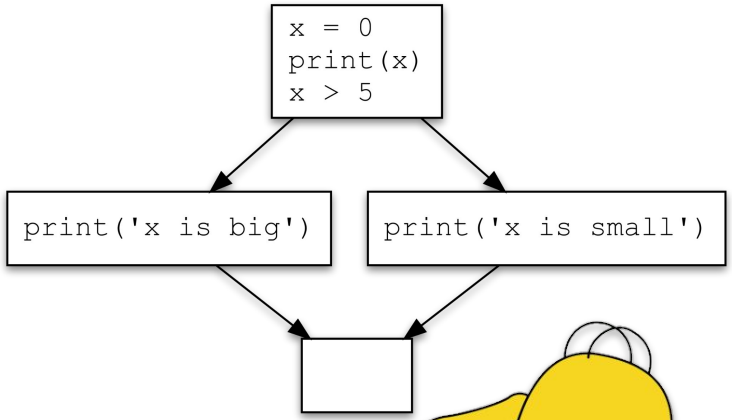
- **What is it?**
  - How execution flows from one application component to others
- **Why do we care?**
  - ???

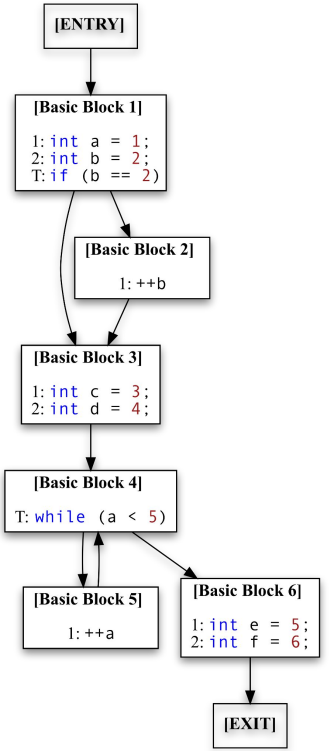




Control Flow

- **What is it?**
  - How execution flows from one application component to others
- **Why do we care?**
  - Want to understand the entire program!





Recovering Control Flow

- **Direct Edges**
  - Jump/call a function

```
jmp 0x4001AB3
```

Target is pre-set **statically**

Recovering Control Flow

- **Direct Edges**
  - Jump/call a function
- **Indirect Edges**
  - Transfer to a register
  - Function pointers
  - Switch-case tables

```
jmp 0x4001AB3
```

Target is pre-set **statically**

```
call %eax; where?
```

Target found at **runtime**

Recovering Control Flow

- **Direct Edges**
  - Jump/call a function
- **Indirect Edges**
  - Transfer to a register
  - Function pointers
  - Switch-case tables
- **"Pseudo" Edges**
  - Post-call returns

`jmp 0x4001AB3`

Target is pre-set **statically**

`call %eax;` where?

Target found at **runtime**

`ret;` goes where?

Necessary to recover **all paths**

Recovering Control Flow

- **Direct Edges**
  - Jump/call a function

```
jmp 0x4001AB3
```

Target is pre-set **statically**

- **Indirect Edges**
  - Transfer to a register
  - Function pointers
  - Switch-case tables

```
call %eax; where?
```

Target found at **runtime**

- **"Pseudo" Edges**
  - Post-call returns

```
ret; goes where?
```

Necessary to recover **all paths**

- **Tail Calls**
  - Call at function's end

```
jmp &foo; call?
```

Expressed as **jumps**, not calls

Symbol Stripping

- **Debugging symbols:** maps instructions in the compiled binary program to their corresponding **variable**, **function**, or **line** in the source code.

```
int addition(int num1, int num2){
     return num1+num2;
}

int main(){
     int var1, var2;
     printf("Enter number 1: ");
     scanf("%d",&var1);
     printf("Enter number 2: ");
     scanf("%d",&var2);
     int res = addition(var1, var2);
     printf ("Output: %d", res);
     return 0;
}
```

Symbol Stripping

- **Debugging symbols:** maps instructions in the compiled binary program to their corresponding **variable**, **function**, or **line** in the source code.
  - Makes **RE easy** if you have symbols...

```c
int addition(int num1, int num2){
     return num1+num2;
}

int main(){
     int var1, var2;
     printf("Enter number 1: ");
     scanf("%d",&var1);
     printf("Enter number 2: ");
     scanf("%d",&var2);
     int res = addition(var1, var2);
     printf ("Output: %d", res);
     return 0;
}
```

```
$ objdump --syms example | grep .text
0000000000001090 l F .text 0000000000000000 deregister_tm_clones
00000000000010c0 l F .text 0000000000000000 register_tm_clones
0000000000001100 l F .text 0000000000000000 __do_global_dtors_aux
0000000000001140 l F .text 0000000000000000 frame_dummy
0000000000001150 g F .text 0000000000000018 addition
0000000000001060 g F .text 0000000000000026 _start
0000000000001170 g F .text 0000000000000085 main
```

Symbol Stripping

- **Debugging symbols:** maps instructions in the compiled binary program to their corresponding **variable**, **function**, or **line** in the source code.
  - Makes **RE easy** if you have symbols… but **often stripped** from the binary!

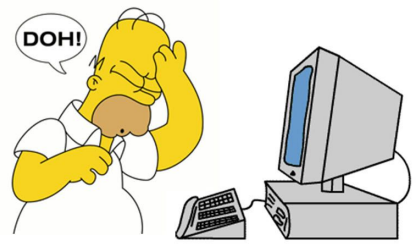
```c
int addition(int num1, int num2){
     return num1+num2;
}

int main(){
     int var1, var2;
     printf("Enter number 1: ");
     scanf("%d",&var1);
     printf("Enter number 2: ");
     scanf("%d",&var2);
     int res = addition(var1, var2);
     printf ("Output: %d", res);
     return 0;
}
```

```
$ objdump --syms example

example:     file format elf64-x86-64

SYMBOL TABLE:
no symbols
```



Obfuscation

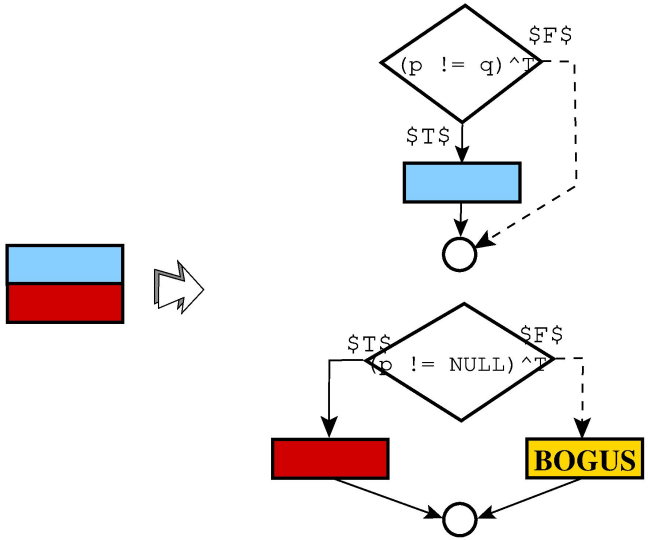
- **Obfuscation:** techniques designed to make third-party analysis **difficult**
  - **???**

Obfuscation

- **Obfuscation:** techniques designed to make third-party analysis **difficult**
  - Developers want to keep their intellectual property secret to just themselves!

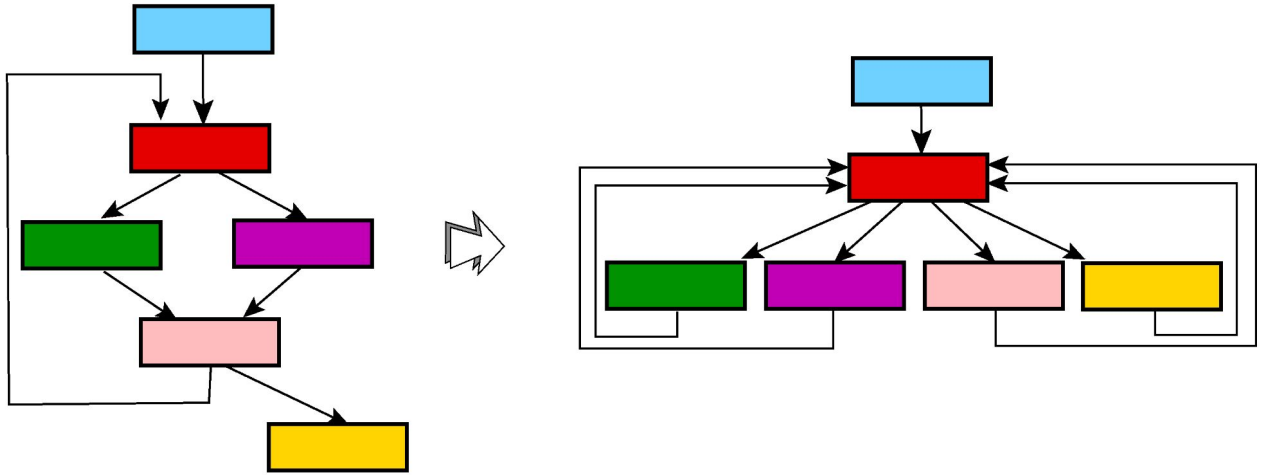
Obfuscation

- **Obfuscation:** techniques designed to make third-party analysis **difficult**
  - Developers want to keep their intellectual property secret to just themselves!
  - **Example: opaque predicates** → introduces “fake” control-flow that is confusing!



Obfuscation

- **Obfuscation:** techniques designed to make third-party analysis **difficult**
  - Developers want to keep their intellectual property secret to just themselves!
  - **Example: control-flow flattening** → removes any recognizable flow ordering



Questions?

Pillars of RE: Structure Recovery

Program Structure

- **Why do we care?**
  - ???

Program Structure

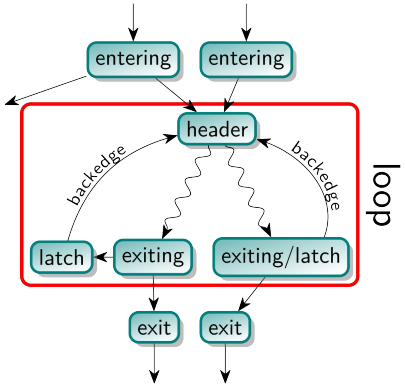
- **Why do we care?**
  - Know how the code's parts work together

```
dec edi
push byte +0x1e
mov bh,0xb5
or al,0x12
push byte +0x4
jmp dword 0xb12b:0x7ba58ea0
mov cl,0x14
```

Program Structure

- **Why do we care?**
  - Know how the code's parts work together
- **Examples:**
  - Basic Blocks
  - Loop Types
  - Recursion
  - Jump Tables
  - Functions

```
dec edi
push byte +0x1e
mov bh,0xb5
or al,0x12
push byte +0x4
jmp dword 0xb12b:0x7ba58ea0
mov cl,0x14
```

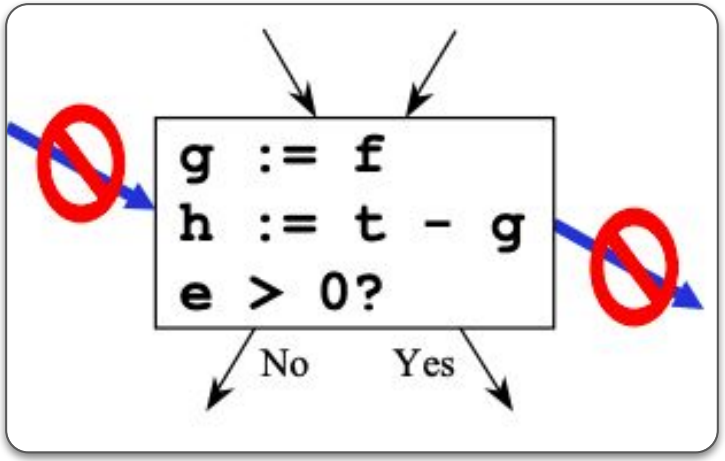


Structure Recovery

- Largely **heuristic**-based
  - Construct-specific rules

Structure Recovery

- Largely **heuristic**-based
  - Construct-specific rules
- **Basic Blocks:**
  - **Start:**
    - Target of a `jmp`
    - Target of a `call`
    - Target of a `ret`
  - **End:**
    - Ends in a `jmp`
    - Ends in a `call`
    - Ends in a `ret`



Structure Recovery

- Largely **heuristic**-based
  - Construct-specific rules
- **Functions:**
  - **Start:**
    - Target of a `call`
    - Target of a tail call
    - A known prologue
    - A dispatch table entry
  - **End:**
    - Location of a `ret`
    - Location of a tail call
    - A known epilogue

```
push ebp
mov ebp, esp
sub esp, N
```

Prologue

```
mov esp, ebp
pop ebp
ret
```

Epilogue

```
switch(choice) {
    case 0 :
        result = add(first, second);
        break;
    case 1 :
        result = sub(first, second);
        break;
    case 2 :
        result = mult(first, second);
        break;
    case 3 :
        result = divide(first, second);
        break;
}
```

C-level Switch Table

Questions?

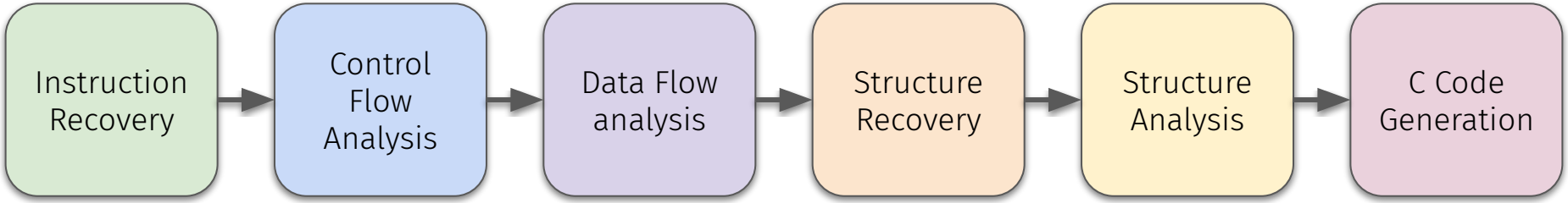
RE Tasks: Decompilation

Decompilation

- **Goal: ???**

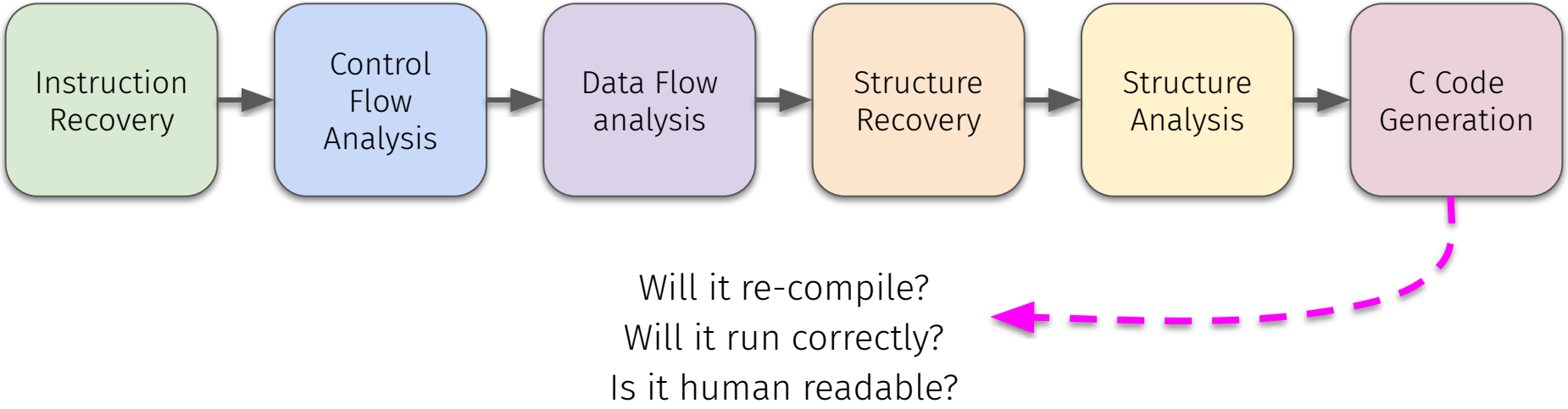
Decompilation

- **Goal:** obtain **semantically-equivalent** source code from a compiled binary



Decompilation

- **Goal:** obtain **semantically-equivalent** source code from a compiled binary
  - **In practice: really difficult** with little guarantee of success (**compilable** or **correct** code)



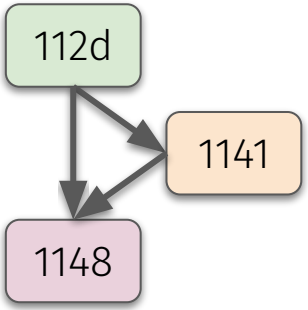
Try it yourself!

```
112d: push %ebp
112e: mov %esp,%ebp   // mov src,dst
1131: mov %edi,$0x14(%ebp)
1134: mov $0x0,$0x4(%ebp)
113b: cmp $0x1,$0x14(%ebp)
113f: jne 1148
1141: add 0x1337,$0x4(%ebp)
1148: mov $0x0,%eax
114d: pop %ebp
114e: ret
114f: nop
```

https://rev.fish/files/bar2022_keynote.pdf

Try it yourself!

```
112d: push %ebp
112e: mov %esp,%ebp    // mov src,dst
1131: mov %edi,$0x14(%ebp)
1134: mov $0x0,$0x4(%ebp)
113b: cmp $0x1,$0x14(%ebp)
113f: jne 1148
1141: add 0x1337,$0x4(%ebp)
1148: mov $0x0,%eax
114d: pop %ebp
114e: ret
114f: nop
```



Variables:

ebp-0x4:	foo
ebp-0x14:	bar

```
foo = 0; // 1134
if (bar == 1) { // 113b
     // 1141
     foo = foo + 0x1337;
}
return 0; // 1148
```

https://rev.fish/files/bar2022_keynote.pdf

Popular Decompilers

- **Many decompilers** available today (both commercial and open-source)
  - Can lift binaries to **different languages** (e.g., C/C++, LLVM IR, custom IRs, etc.)

angr

IDA Pro

Binary Ninja

Ghidra

Different Decompilers = Different Outputs

- **Example:** `HelloWorld` (ARM version) on DogBolt.org

angr 9.2.38

```
1  int _init()
2  {
3      unsigned int v0;  // [bp-0x8]
4      unsigned int v1;  // [bp-0x4]
5      unsigned int v2;  // lr
6      unsigned int v3;  // r3
7
8      v1 = v2;
9      v0 = v3;
10     return call_weak_fn();
11 }
12
13 int _start(unsigned int a0)
14 {
15     unsigned int v0;  // [bp-0x8]
16     unsigned int v1;  // [bp-0x4]
17     unsigned int v2;  // [bp+0x0]
18
19     v2 = stack_base + 4;
20     v1 = a0;
21     v0 = 0;
22     __libc_start_main(); /* do not return */
23 }
24
25 int sub_10390()
26 {
27     abort(); /* do not return */
28 }
29
30 int call_weak_fn()
31 {
32     return;
33     __gmon_start__();
34     return;
35 }
36
37 int dereaister_tm_clones()
```

BinaryNinja 3.3.3996 (e34a955e)

```
1  int32_t _init(int32_t arg1, int32_t arg2)
2  {
3      return call_weak_fn(arg1, arg2);
4  }
5
6  int32_t sub_10308()
7  {
8      /* jump -> 0 */
9  }
10
11 void __libc_start_main(
12     int32_t (* main)(int32_t argc, char** argv, char
13     char** ubp_av, void (* init)(), void (* fini)(),
14     void* stack_end) __noreturn
15 {
16     /* tailcall */
17     return __libc_start_main(main, argc, ubp_av, ini
18 }
19
20 int32_t getchar()
21 {
22     /* tailcall */
23     return getchar();
24 }
25
26 int32_t puts(char const* str)
27 {
28     /* tailcall */
29     return puts(str);
30 }
31
32 int32_t __gmon_start__()
33 {
34     /* tailcall */
35     return __gmon_start__();
36 }
37
```

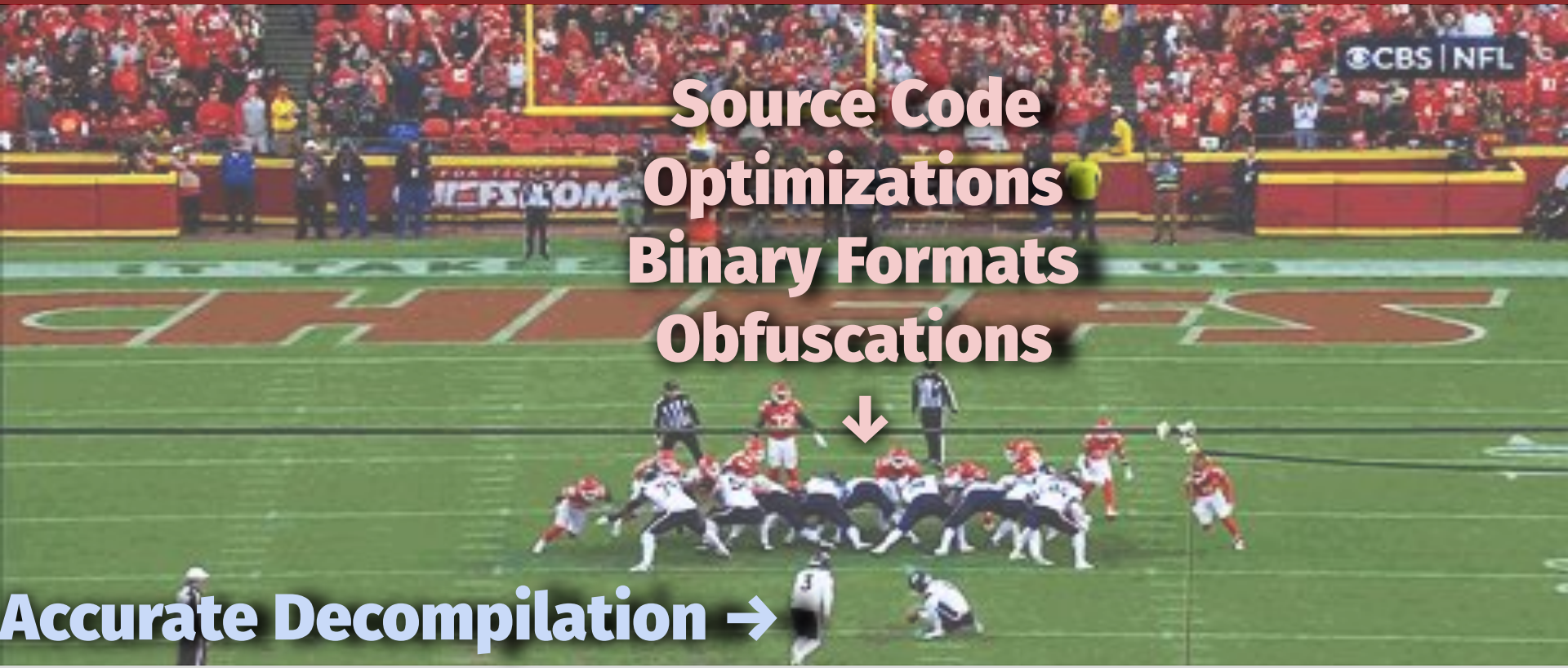
Ghidra 10.2.2 (9813cde2)

```
1  #include "out.h"
2
3
4
5  int _init(EVP_PKEY_CTX *ctx)
6
7  {
8    int iVar1;
9
10   iVar1 = call_weak_fn();
11   return iVar1;
12 }
13
14
15
16 void __libc_start_main(void)
17
18 {
19   __libc_start_main();
20   return;
21 }
22
23
24
25 // WARNING: Unknown calling convention -- yet parame
26
27 int getchar(void)
28
29 {
30   int iVar1;
31
32   iVar1 = getchar();
33   return iVar1;
34 }
35
36
37
```

Hex-Rays 8.2.0.221215

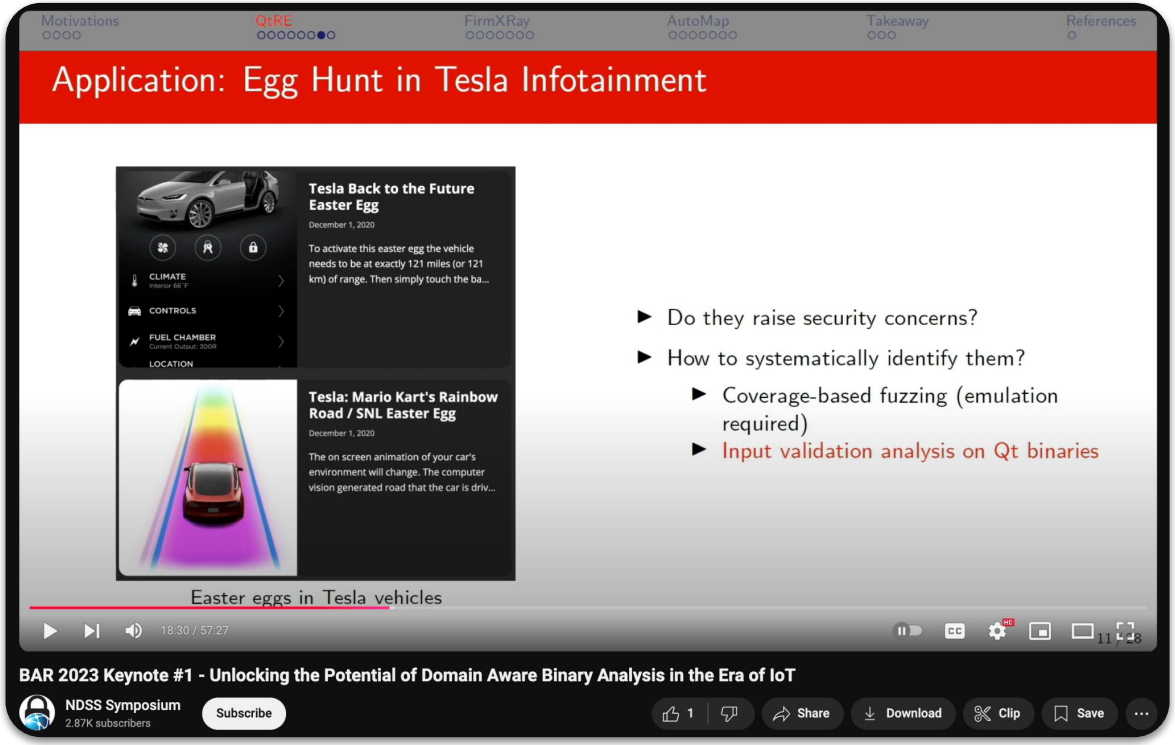
```
1  /* This file was generated by the Hex-Rays decompile
2     Copyright (c) 2007-2021 Hex-Rays <info@hex-rays.c
3
4     Detected compiler: GNU C++
5  */
6
7  #include <defs.h>
8
9  #include <stdarg.h>
10
11
12 //-------------------------------------------------
13 // Function declarations
14
15 int init_proc();
16 void sub_10308();
17 // int __fastcall _libc_start_main(int (__fastcall *
18 // int getchar(void);
19 // int puts(const char *s);
20 // int _gmon_start__(void); weak
21 // void abort(void);
22 void __noreturn start(void (*a1)(void), int a2, int
23 int call_weak_fn();
24 char *deregister_tm_clones();
25 __int64 register_tm_clones();
26 char *_do_global_dtors_aux();
27 int __cdecl main(int argc, const char **argv, const
28 void term_proc();
29
30 //-------------------------------------------------
31 // Data declarations
32
33 char _bss_start; // weak
34 // extern _UNKNOWN __gmon_start__; weak
35
36
37 //----- (000102FC) --------------------------------
```

Challenges to Binary Decompilation



Supplemental Content: Domain-specific RE

- Dr. Zhiqiang Lin's keynote at BAR'23
- **Lots of cool bugs!**
  - Tesla Infotainment
  - "Super Apps"
  - And more!
- **Check it out!**



Next time on CS 4440...

Attacking Large Language Models (guest lecture)