

CSE227 - Graduate Computer Security

Course introduction, definitions, reflections on trusting trust

UC San Diego

whoami



undergrad



grad



postdoc

Deepak Kumar
Assistant Professor in CSE

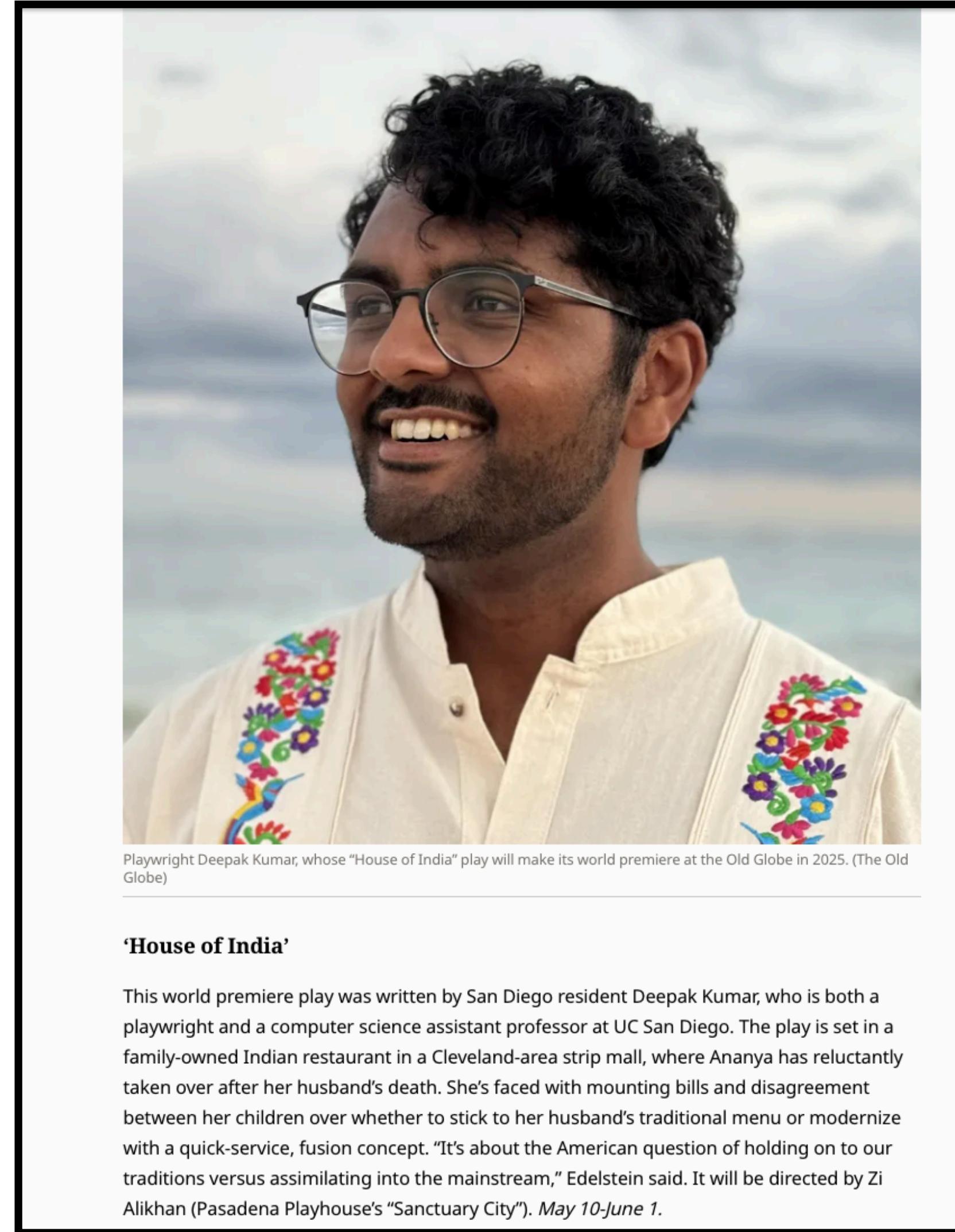
Background on me (research)

- I work primarily on applied computer security research, my research interests are in *sociotechnical security* (computer security + HCI)
 - Interests primarily in ways that technology + society interact, and where security or safety problems arise (e.g., online harassment, mis/disinformation, AI generated deepfakes, trust on the Internet, etc.)
- Mostly, I'm a data + systems guy.... e.g.,
 - "What does the marketplace for nonconsensual sexual deepfake creation look like?"
 - "How much toxic content there on Reddit, and what can we learn about attack patterns than inform defenses?"
 - "How can we build better defenses for journalists facing online harassment on social media?"

How'd I get into security?

- Two versions (both are true)
 - I've always been very interested in technology + society, and computer security is a field that by definition gets to impact both of those interests
 - I wanted to do computer architecture research in undergrad, but I didn't get a good enough grade, but a security group was looking for students and I got an A, so...
- Started in network security, moved my way towards more human-centered work

I also do other things...



What are we (UCSD) known for in security?

- Measurement (cybercrime, malware, spam, captchas, fraud, etc.)
- Defenses (threat intelligence, cyber hygiene, etc.)
- Embedded Security (hacking cars, voting machines, airplanes, credit card skimmers, medical devices)
- Web security + PLsec (cookies, information flow, wasm runtime shenanigans)
- Intersection of crypto + security (implementing crypto is very hard)
- Lots of faculty here working on stuff
 - <https://cryptosec.ucsd.edu>
 - <http://sysnet.ucsd.edu>

Course Staff



Deepak Kumar
Assistant Professor, CSE
kumarde@ucsd.edu



Tianyi Shan
PhD Student, CSE
tshan@ucsd.edu

5-Minute Introductions

- Find two other people in the classroom you're sitting nearby (ideally people you haven't met)
- Take a selfie of your group and email it to us (one per group is fine). Follow directions:
 - Names, degree programs + year progress (e.g., 2nd year MS student)
 - What brought you to the class / what interests you?
 - Something you do for fun
- *Email* written introductions to
cse227-wi25-g@ucsd.edu



Let's recap the last 5-minutes of your life

Let's recap the last 5-minutes of your life

- Some guy (me) who most of you just met stood up and claimed to be the professor of the class

Let's recap the last 5-minutes of your life

- Some guy (me) who most of you just met stood up and claimed to be the professor of the class
- He asked you to do a task that required some effort (and therefore took up your time)

Let's recap the last 5-minutes of your life

- Some guy (me) who most of you just met stood up and claimed to be the professor of the class
- He asked you to do a task that required some effort (and therefore took up your time)
- He asked you to send a *compilation of information about yourselves* to a random email address

Let's recap the last 5-minutes of your life

- Some guy (me) who most of you just met stood up and claimed to be the professor of the class
- He asked you to do a task that required some effort (and therefore took up your time)
- He asked you to send a *compilation of information about yourselves* to a random email address

Why did you do that?

Security is all about trust

You can't have security if you trust no one

Security is all about trust

You can't have security if you trust no one

- With those same groups (no tricks this time), answer the following questions:
 - What is security?
 - What is computer security?
 - What is *trust*?

Security is all about trust

You can't have security if you trust no one

- With those same groups (no tricks this time), answer the following questions:
 - **What is security?**
 - What is computer security?
 - What is *trust*?

Definitions: Security

- Merriam-Webster online dictionary:
 - The quality or state of being secure: such as
 - Free from *danger* : safety
 - Freedom from fear or anxiety
 - Freedom for the prospect of being laid off (job security)

Security is all about trust

You can't have security if you trust no one

- With those same groups (no tricks this time), answer the following questions:
 - What is security?
 - **What is computer security?**
 - What is trust?

Definitions: Computer Security

- Most of computer science is about **functionality**:
 - UX/UI
 - Architecture
 - AI / ML development
 - Operating Systems / Networking / Databases
 - Compilers / PL
 - Microarchitecture
- Computer security is **not** about functionality
- Computer security is the ***study of a computer system in the presence of an adversary***
- Holistic property:
 - “Software security is about integrating security practices into the way you build software, not integrating security features into your code” – Gary McGraw”

Security is all about trust

You can't have security if you trust no one

- With those same groups (no tricks this time), answer the following questions:
 - What is security?
 - What is computer security?
 - **What is trust?**

Security is all about trust

You can't have security if you trust no one



trust
/trəst/

noun

1. firm belief in the reliability, truth, ability, or strength of someone or something.
"relations have to be built on trust"

Similar:

confidence

belief

faith

freedom from suspicion/doubt

sureness



2. LAW

an arrangement whereby a person (a trustee) holds property as its nominal owner for the good of one or more beneficiaries.

"a trust was set up"

Similar:

safe keeping

keeping

protection

charge

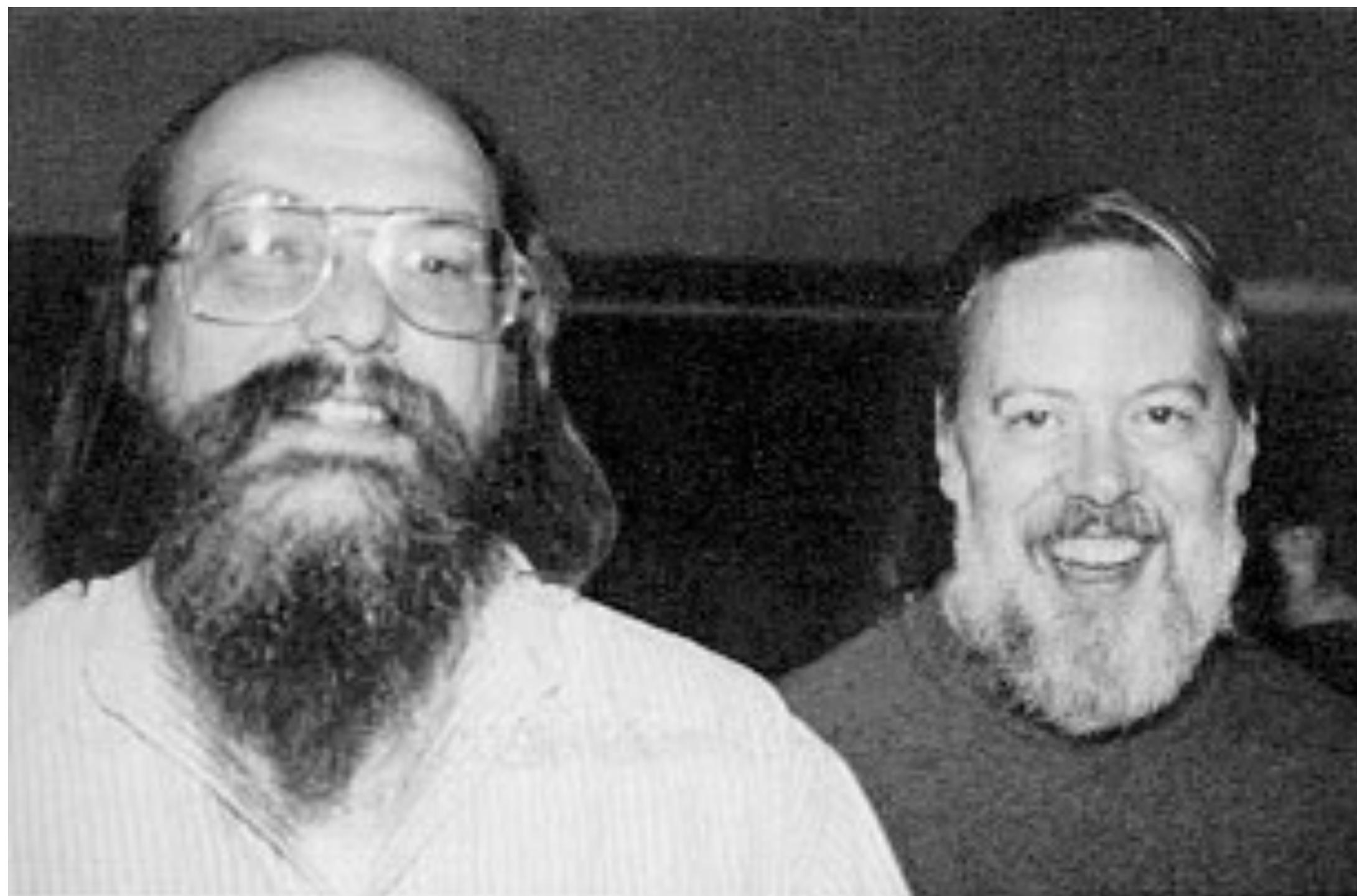
care

custody



Reflections on Trusting Trust

1984 Turing Award Lecture



Ken Thompson + Dennis Ritchie

TURING AWARD LECTURE

Reflections on Trusting Trust

To what extent should one trust a statement that a program is free of Trojan horses? Perhaps it is more important to trust the people who wrote the software.

KEN THOMPSON

INTRODUCTION

I thank the ACM for this award. I can't help but feel that I am receiving this honor for timing and serendipity as much as technical merit. UNIX¹ swept into popularity with an industry-wide change from central mainframes to autonomous minis. I suspect that Daniel Bobrow [1] would be here instead of me if he could not afford a PDP-10 and had had to "settle" for a PDP-11. Moreover, the current state of UNIX is the result of the labors of a large number of people.

There is an old adage, "Dance with the one that brought you," which means that I should talk about UNIX. I have not worked on mainstream UNIX in many years, yet I continue to get undeserved credit for the work of others. Therefore, I am not going to talk about UNIX, but I want to thank everyone who has contributed.

That brings me to Dennis Ritchie. Our collaboration has been a thing of beauty. In the ten years that we have worked together, I can recall only one case of miscoordination of work. On that occasion, I discovered that we both had written the same 20-line assembly language program. I compared the sources and was astounded to find that they matched character-for-character. The result of our work together has been far greater than the work that we each contributed.

I am a programmer. On my 1040 form, that is what I put down as my occupation. As a programmer, I write

programs. I would like to present to you the cutest program I ever wrote. I will do this in three stages and try to bring it together at the end.

STAGE I

In college, before video games, we would amuse ourselves by posing programming exercises. One of the favorites was to write the shortest self-reproducing program. Since this is an exercise divorced from reality, the usual vehicle was FORTRAN. Actually, FORTRAN was the language of choice for the same reason that three-legged races are popular.

More precisely stated, the problem is to write a source program that, when compiled and executed, will produce as output an exact copy of its source. If you have never done this, I urge you to try it on your own. The discovery of how to do it is a revelation that far surpasses any benefit obtained by being told how to do it. The part about "shortest" was just an incentive to demonstrate skill and determine a winner.

Figure 1 shows a self-reproducing program in the C³ programming language. (The purist will note that the program is not precisely a self-reproducing program, but will produce a self-reproducing program.) This entry is much too large to win a prize, but it demonstrates the technique and has two important properties that I need to complete my story: 1) This program can be easily written by another program. 2) This program can contain an arbitrary amount of excess baggage that will be reproduced along with the main algorithm. In the example, even the comment is reproduced.

¹UNIX is a trademark of AT&T Bell Laboratories.

© 1984 0001-0782/84/0800-0761 75¢

Reflections on Trusting Trust

How do we run C programs?

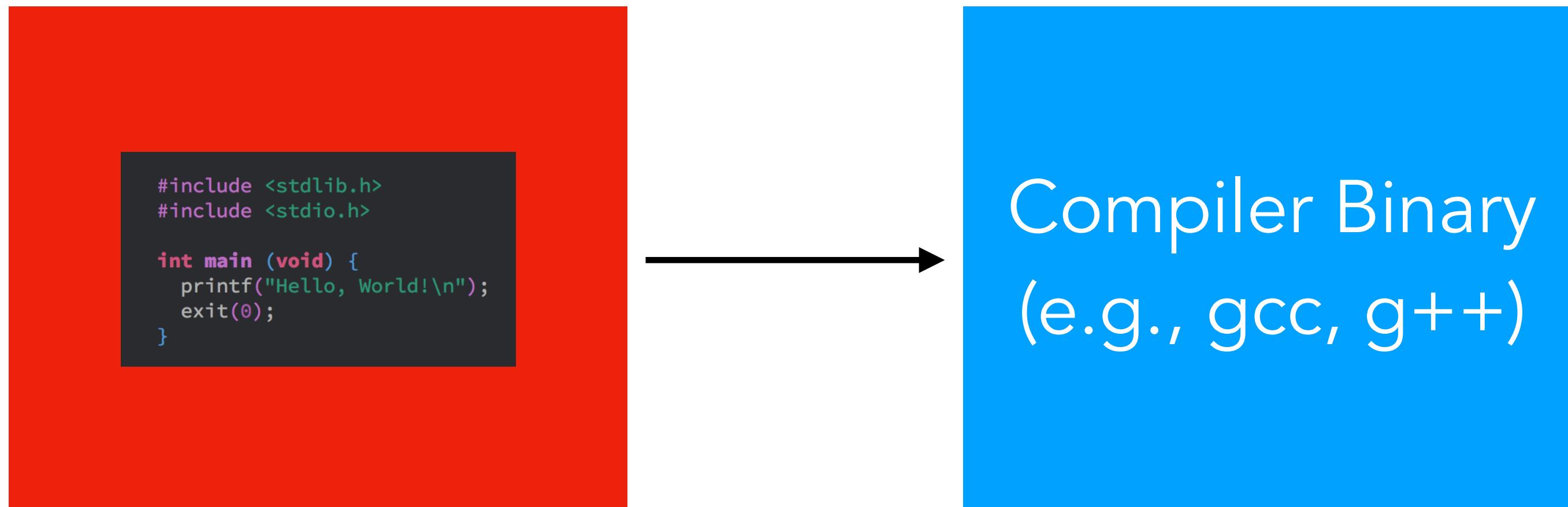
C Program

```
#include <stdlib.h>
#include <stdio.h>

int main (void) {
    printf("Hello, World!\n");
    exit(0);
}
```

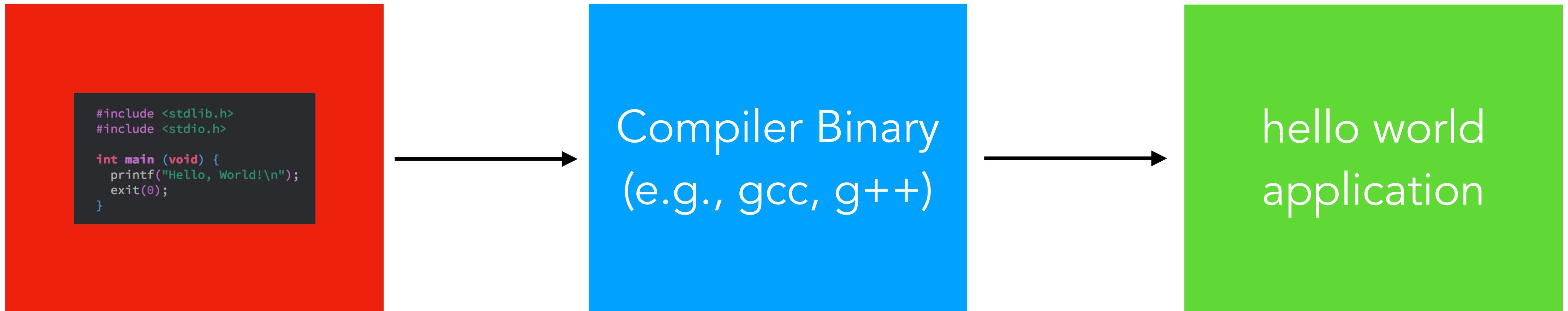
Reflections on Trusting Trust

How do we run C programs?



Reflections on Trusting Trust

How do we run C programs?

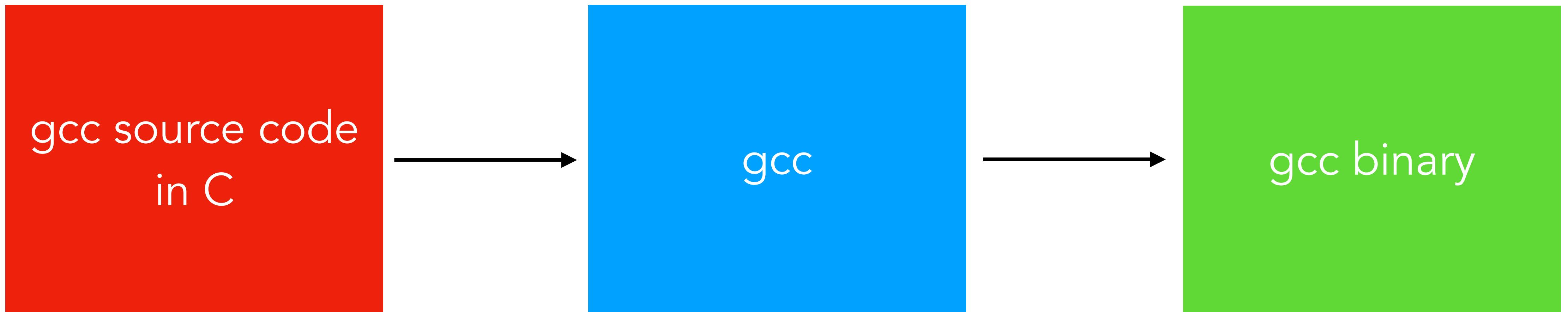


What language is gcc written in?

What language is gcc written in?

Surprise! It's C.

gcc compiles gcc



gcc compiles gcc

Simple function for parsing and escaping characters

gcc source code
in C



```
...
c = next( );
if(c != '\\\\')
    return(c);
c = next( );
if(c == '\\\\')
    return('\\\\');
if(c == 'n')
    return('\\n');
...
...
```

gcc compiles gcc

Let's say I wanted to add '\v' – how would I do that?

gcc source code
in C



```
...
c = next( );
if(c != '\\\\')
    return(c);
c = next( );
if(c == '\\\\')
    return('\\\\');
if(c == 'n')
    return('\\n');
...
...
```

gcc compiles gcc

Let's say I wanted to add '\v' – how would I do that?

gcc source code
in C



```
...
c = next( );
if(c != '\\\\')
    return(c);
c = next( );
if(c == '\\\\')
    return('\\\\');
if(c == 'n')
    return('\\n');
if(c == 'v')
    return('\\v');
...

```

gcc compiles gcc

Let's say I wanted to add '\v' – how would I do that?

gcc source code
in C



```
...
c = next( );
if(c != '\\\\')
    return(c);
c = next( );
if(c == '\\\\')
    return('\\\\');
if(c == 'n')
    return('\\n');
if(c == 'v')
    return('\\v');
...
...
```

This will throw a compilation error.
Why?

gcc compiles gcc

Let's say I wanted to add '\v' – how would I do that?

gcc source code
in C



```
...
c = next( );
if(c != '\\')
    return(c);
c = next( );
if(c == '\\')
    return('\\');
if(c == 'n')
    return('\n');
if(c == 'v')
    return(11);
...
```

We have to tell the C compiler about '\v' before
we can use '\v'

gcc compiles gcc

Let's say I wanted to add '\v' – how would I do that?

gcc source code
in C



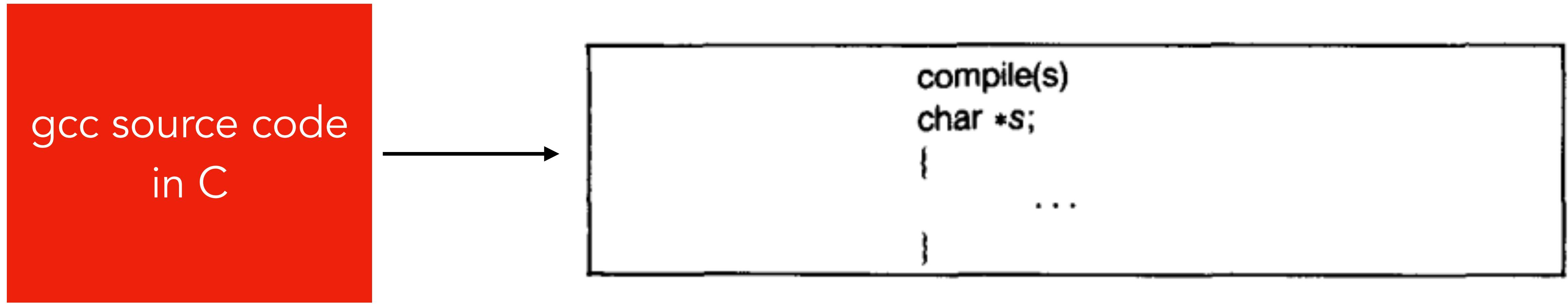
```
...
c = next( );
if(c != '\\\\')
    return(c);
c = next( );
if(c == '\\\\')
    return('\\\\');
if(c == 'n')
    return('\\n');
if(c == 'v')
    return('\\v');
...

```

Now, this will compile!

gcc can compile bad gcc

Trojan Horsing the C compiler to bug code



compile(s) compiles the next line of source code

gcc can compile bad gcc

Trojan Horsing the C compiler to bug code

gcc source code
in C



```
compile(s)
char *s;
{
    if(match(s, "pattern")) {
        compile("bug");
        return;
    }
    ...
}
```

If I owned the C compiler, I could add logic that introduces malicious bugs when certain patterns appear

gcc can compile bad gcc

Trojan Horsing the C compiler to bug code



gcc can compile bad gcc

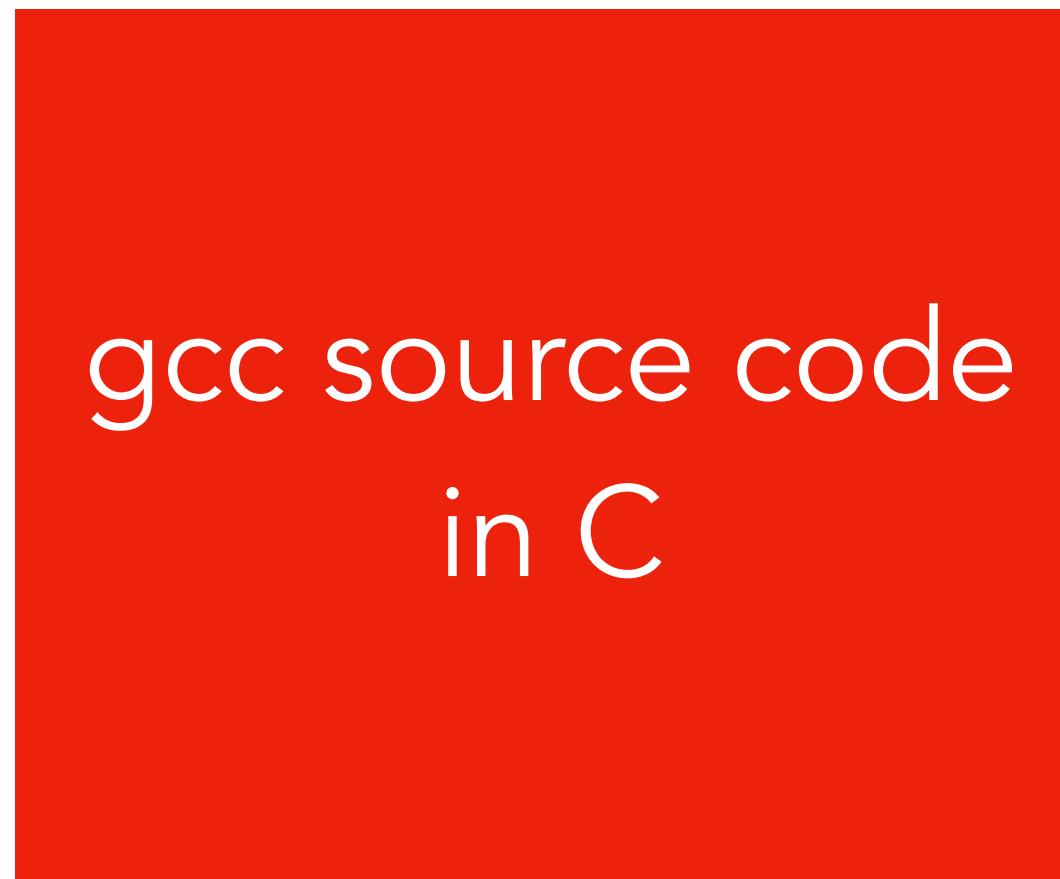
Trojan Horsing the C compiler to bug code



This is very easy to detect if you read the C compiler code.

gcc can compile bad gcc without you knowing it's compiling bad gcc

Trojan horsing the C compiler to bug code without you knowing it's trojan horsed



```
compile(s)
char *s;
{
    if(match(s, "pattern1")) {
        compile ("bug1");
        return;
    }
    if(match(s, "pattern 2")) {
        compile ("bug 2");
        return;
    }
    ...
}
```

In addition to matching *login*, I could also match the *C compiler itself*, and compile in both trojans

So then, the following can happen...

- Install the double-trojaned version as the *compiler* on a machine
- You can change the source code of C back to the non-malicious version
- Anytime someone compiles *login* or even the compiler, they'll still get the *login* bug even when it's nowhere to be found in the source code
- **Moral: You can't trust code that you did not totally create yourself (so, everything)**
 - Compilers, assemblers, loaders, even hardware microcode
 - Deepak's extension: You can't have security without **trust**

This *actually* happens in real life

XZ Utils Backdoor – CVE-2024-3094

- In February 2024, a Microsoft employee found a backdoor in XZ, a popular compressor for Unix-like operating systems
 - The backdoor was found in compressed test files that enabled **full remote-code execution on the machine using the compromised xz** via sshd
 - The backdoor was introduced *in the supply chain* – a developer (Jia Tan) who contributed to XZ for years introduced the patch in **compressed test files**
 - Guess is nation-state, but we're not sure
 - Do you trust that every piece of software on your machine has not been tampered with? Why?

My take on computer security

- Security is not a “*thing you do*”, it is a *way of life*
 - We call this the “adversarial mindset:” how do you think like an attacker so you can be ready to 1) find problems or 2) fix them before they happen?
 - Some paranoia is good, maximal paranoia is worthless; we want a sweet spot, called **rational paranoia**
- Security is always in relation to the threat
 - What does it mean to “be secure?” Against what? With what assumptions?

Course Ethos + Logistics

Course info

- Website: <https://kumarde.com/cse227-wi25/>
- Canvas: <https://canvas.ucsd.edu/courses/61827>
 - Gradescope for grading
 - Piazza for communication (email is fine too)
- Office Hours
 - Tuesday 2 – 3pm (or by appointment), CSE 3248

Goals and non-goals of this course

This is a research + discussion oriented course

- **Goals**

- Learn how to **read papers**, not just for content but **context** (why does this paper exist?)
- Learn how to *talk* about papers and discuss them at a graduate level
- Explore range of current problems + tensions in computer security
- Get feet wet in security research (term project)

- **Non-goals**

- Review of all security mechanisms (go read a book)
- Deep examination of cryptography (there are other courses for you)
- A deep dive into any one subarea (this is a breadth course)

Topics in this course

This is a breadth course

- Software security
- Cyberphysical systems + IoT devices
- Web security
- Network security
- Spam, crime
- Usability + human factors
- Security & society

Structure of this course

- Attendance (5% of your grade)
- Participation (20% of your grade)
- Term Project (75% of your grade)

Attendance

- Attendance is **mandatory** – it will be checked every class
 - You can miss up to 2 classes without telling me, exceptions on case by case basis

Participation

- There will be assigned readings every session starting 1/14
 - You must have read and be prepared to discuss the material at the beginning of class
 - The *entire* course will be structured like a discussion-forward lecture led by me
 - Each session, I will “cold call,” which are random calls to students to answer questions about the paper and begin discussing the paper
 - Questions are a mix of comprehension + discussion
 - You get 3 *skips* throughout the quarter, but expect to be called on at least 10 times during the quarter

Term Project

- You will form groups of 3 – 4 and work on an original research project over the course of the term in **computer security**
 - Intention is to give you experience conducting research in this area
- Goal should be rigorous, original research, with ideally **publishable** results
 - Many similar projects from 227 have gone on to be published at conferences, win awards, etc...
- If you are already doing research broadly in this area, **talk to me to see if it will fit for the requirements of this course.**
- Start talking to each other now! **Teams and initial project idea will be due 1/17**

Term Project – Timeline

- Project spec will be released by 1/10
- Milestones
 - Team declaration + Project intention form – Due **1/17 by 11:59pm**
 - Midpoint Check-in – Due **2/14 by 12:30pm**
 - Final Presentation – Last week of the quarter
 - Final writeup – Due **3/18 Anywhere-on-Earth**

Term Project – Checkins

- You can meet with me anytime to talk about your progress, solicit feedback, and chat about ideas
- I will **mandatorily** meet with all teams week 7 to check in on your progress
(2/17 – 2/19)
 - We'll go over the midpoint check-in document, talk about progress and blockers
- More details to come in the project spec

Course Vibes

Community-centric learning

- The classroom is *community*
 - Get to know one another! The course, like your project, is a group effort
 - When you come **prepared**, you're not just doing it for yourself, you're doing it for the good of everyone
 - Your learning is multiplied by your preparedness and others' preparedness
 - Discussions should be *respectful*, understanding everyone is here to contribute and to learn

Next time...

- We'll talk about security research, *why* we do security research, the styles of security research, and discuss some potential directions for projects
- By 1/17
 - Fill out "First Day Survey" <https://canvas.ucsd.edu/courses/61827/quizzes/199237>
 - Project intention document due for term project (group selection + general research idea), more details on submission to come
 - No reading for next time, but be prepared for cold calls starting 1/14