Systems Programming

Computer Operating Systems, Fall 2023

Instructor: Travis McGaha

Head TAs: Nate Hoaglund & Seungmin Han

TAs:

Andy Jiang	Haoyun Qin	Kevin Bernat	Ryoma Harris
Audrey Yang	Jason hom	Leon Hertzberg	Shyam Mehta
August Fu	Jeff Yang	Maxi Liu	Tina Kokoshvili
Daniel Da	Jerry Wang	Ria Sharma	Zhiyan Lu
Ernest Ng	Jinghao Zhang	Rohan Verma	

Administrivia

- PennOS was due Wed Nov 29
  - You will schedule a time to meet with your TA to demonstrate your working code
  - Can submit via gradescope now
  - Reach out to TA's to schedule PennOS Demo

- Check-in due after Midterm 1

pollev.com/tqm

- Any questions, comments or concerns from last lecture?

Lecture Outline

- Systems Programming
- C & C++
- Safety
- What's Next?

pollev.com/tqm

- On a scale of 1 (hate) to 5 (love), how do you feel about C as a programming language?

pollev.com/tqm

- Why do you think we chose C as the programming language for this course?

pollev.com/tqm

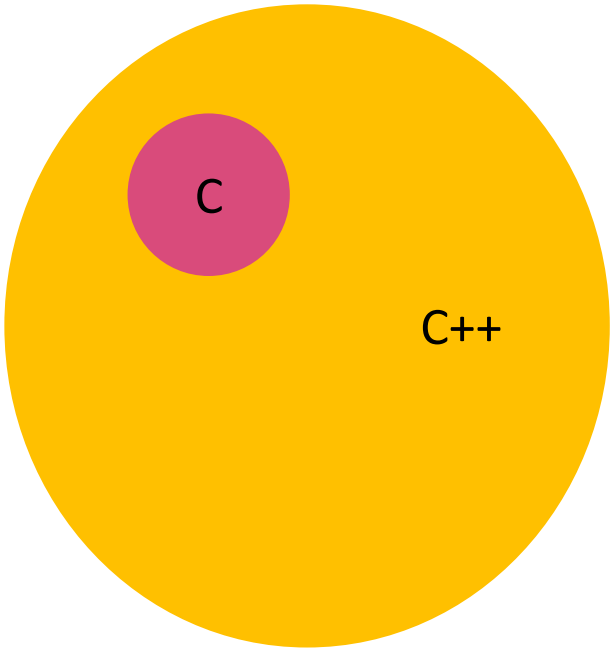
- Why do you think we chose C as the programming language for this course?

- What comes to my mind:
  - C is fast
  - C exposes you to the low-level features that other languages abstract away. (Even if we did not use them all)
    - addresses
    - Memory management
    - System Calls
    - Assembly
  - Operating System Kernels and Systems have been written in C for a long time. In some ways it would be blasphemous to choose something like python

C/C++?

- Common way of listing the languages: C/C++

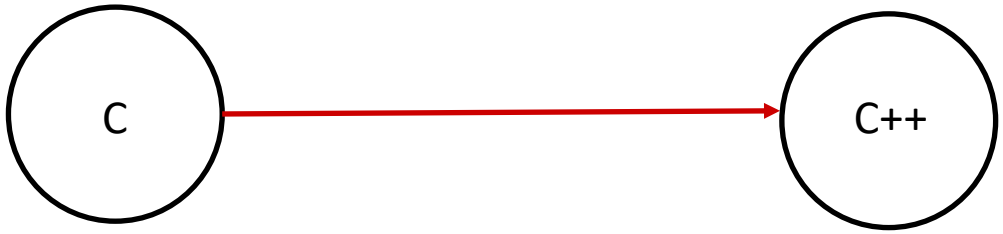
- Common understanding of the language
  - C++ is C but more
  - C++ is a super set of C



- This understanding is a pet-peeve of mine

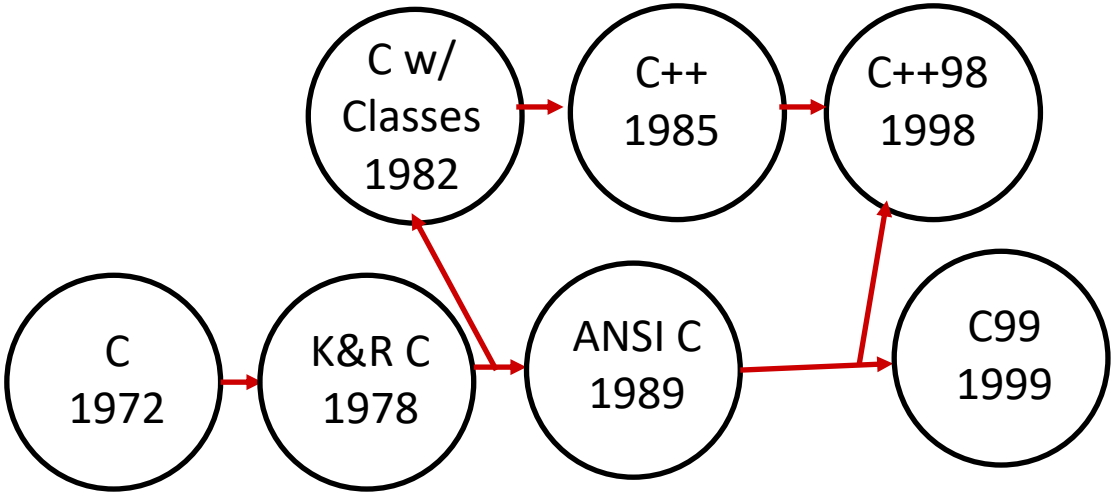
C vs C++ (Timeline)

- What People Think



C vs C++ (Timeline)

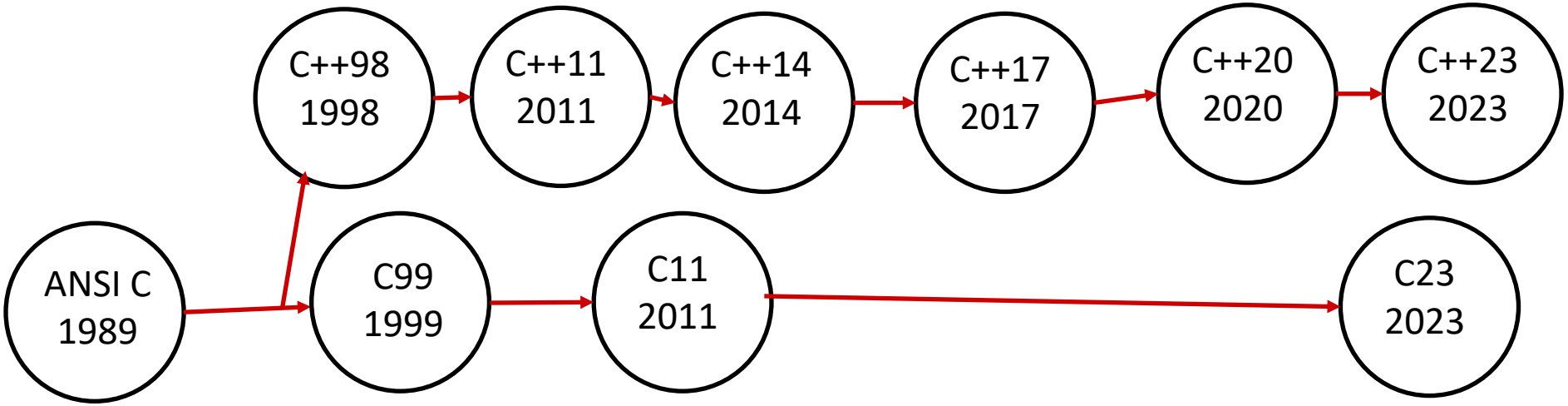
- More Detail (but a lot left out)



THE LANGUAGES "FORK" around 1999
Not all C99 features are legal C++, but most of them are.

C vs C++ (Timeline)

- More Detail (but a lot left out)



THE LANGUAGES "FORK" around 1999
Not all C99 features are legal C++, but most of them are.

C has adopted changes from C++
example: `auto` and `nullptr` in C23

C vs C++ Examples

- old_c.c
  - C has evolved since it was introduced in 1972
- c23.c
  - C still gets updates adding new features
  - Admittedly, the updates are small relative to other language updates
- cpp23.cpp and stdin_echo.cpp
  - Modern C++ is very different from C (Though most C is still legal!)
- cpp23_hello.cpp
  - The fundamentals of the language are changing as well

Learning a little bit of C++

- String & Vector
  - You should be familiar with these, brief demo in learning.cpp
- References
  - Next slide…
  - And in ref.cpp

Poll Everywhere

pollev.com/tqm

- What is printed in this example?

A. Output "3"

B. Output "0"

```
#include <iostream>
#include <vector>

using namespace std;

void foo(vector<int> v) {
  v.push_back(2400);
  v.push_back(5950);
  v.push_back(3800);
}

int main(int argc, char** argv) {
  vector<int> v;
  foo(v);
  cout << v.size() << endl;
}
```

Poll Everywhere

pollev.com/tqm

- What is printed in this example?

A. Output "3"

B. Output "0"

```
#include <iostream>
#include <vector>

using namespace std;

void foo(vector<int> v) {
  v.push_back(2400);
  v.push_back(5950);
  v.push_back(3800);
}

int main(int argc, char** argv) {
  vector<int> v;
  foo(v);
  cout << v.size() << endl;
}
```

References

Note: Arrow points to *next* instruction.

- A **reference** is an alias for another variable
  - *Alias*: another name that is bound to the aliased variable
    - Mutating a reference ***is*** mutating the aliased variable
  - Introduced in C++ as part of the language

```
int main(int argc, char** argv) {
  int x = 5, y = 10;
  int& z = x;

  z += 1;
  x += 1;

  z  = y;
  z += 1;

  return EXIT_SUCCESS;
}
```

When we use '&' in a type declaration, it is a reference.

&var still is "address of var"

x	5

y	10

References

Note: Arrow points to *next* instruction.

- A **reference** is an alias for another variable
  - *Alias*: another name that is bound to the aliased variable
    - Mutating a reference ***is*** mutating the aliased variable
  - Introduced in C++ as part of the language

```
int main(int argc, char** argv) {
  int x = 5, y = 10;
  int& z = x;  // binds the name "z" to x

  z += 1;
  x += 1;

  z  = y;
  z += 1;

  return EXIT_SUCCESS;
}
```


x, z	5

y	10

References

Note: Arrow points to *next* instruction.

- A **reference** is an alias for another variable
  - *Alias*: another name that is bound to the aliased variable
    - Mutating a reference ***is*** mutating the aliased variable
  - Introduced in C++ as part of the language

```
int main(int argc, char** argv) {
  int x = 5, y = 10;
  int& z = x;  // binds the name "z" to x

  z += 1;  // sets z (and x) to 6
  x += 1;

  z  = y;
  z += 1;

  return EXIT_SUCCESS;
}
```

x, z	6

y	10

References

Note: Arrow points to *next* instruction.

- A **reference** is an alias for another variable
  - *Alias*: another name that is bound to the aliased variable
    - Mutating a reference ***is*** mutating the aliased variable
  - Introduced in C++ as part of the language

```
int main(int argc, char** argv) {
  int x = 5, y = 10;
  int& z = x;  // binds the name "z" to x

  z += 1;  // sets z (and x) to 6
  x += 1;  // sets x (and z) to 7

  z  = y;  // Normal assignment
  z += 1;

  return EXIT_SUCCESS;
}
```

x, z	7

y	10

References

Note: Arrow points to *next* instruction.

- A **reference** is an alias for another variable
  - *Alias*: another name that is bound to the aliased variable
    - Mutating a reference ***is*** mutating the aliased variable
  - Introduced in C++ as part of the language

```
int main(int argc, char** argv) {
  int x = 5, y = 10;
  int& z = x;  // binds the name "z" to x

  z += 1;  // sets z (and x) to 6
  x += 1;  // sets x (and z) to 7

  z  = y;  // sets z (and x) to the value of y
  z += 1;

  return EXIT_SUCCESS;
}
```

x, z	10

y	10

References

Note: Arrow points to *next* instruction.

- A **reference** is an alias for another variable
  - *Alias*: another name that is bound to the aliased variable
    - Mutating a reference ***is*** mutating the aliased variable
  - Introduced in C++ as part of the language

```
int main(int argc, char** argv) {
  int x = 5, y = 10;
  int& z = x;  // binds the name "z" to x

  z += 1;  // sets z (and x) to 6
  x += 1;  // sets x (and z) to 7

  z  = y;  // sets z (and x) to the value of y
  z += 1;  // sets z (and x) to 11

  return EXIT_SUCCESS;
}
```

x, z	11

y	10

Pass-By-Reference

Note: Arrow points to *next* instruction.

- C++ allows you to use real pass-by-*reference*
  - Client passes in an argument with normal syntax
    - Function uses reference parameters with normal syntax
    - Modifying a reference parameter modifies the caller's argument!

```
void swap(int& x, int& y) {
  int tmp = x;
  x = y;
  y = tmp;
}

int main(int argc, char** argv) {
  int a = 5, b = 10;

  swap(a, b);
  cout << "a: " << a << "; b: " << b << endl;
  return EXIT_SUCCESS;
}
```

Parameters are attached
To variables provided by caller

(main) **a**	5

(main) **b**	10

Pass-By-Reference

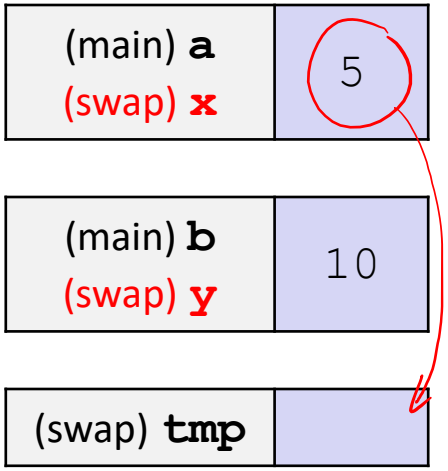
Note: Arrow points to *next* instruction.

- C++ allows you to use real pass-by-*reference*
  - Client passes in an argument with normal syntax
    - Function uses reference parameters with normal syntax
    - Modifying a reference parameter modifies the caller's argument!

```
void swap(int& x, int& y) {
  int tmp = x;
  x = y;
  y = tmp;
}

int main(int argc, char** argv) {
  int a = 5, b = 10;

  swap(a, b);
  cout << "a: " << a << "; b: " << b << endl;
  return EXIT_SUCCESS;
}
```



Pass-By-Reference

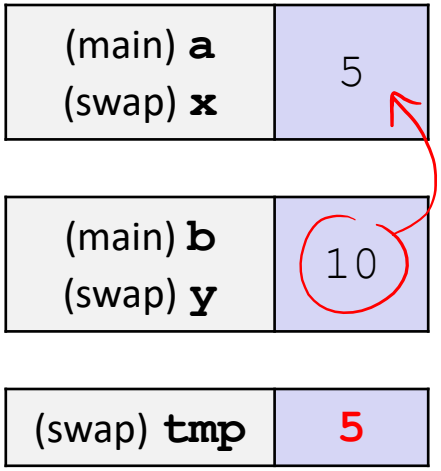
Note: Arrow points to *next* instruction.

- C++ allows you to use real pass-by-*reference*
  - Client passes in an argument with normal syntax
    - Function uses reference parameters with normal syntax
    - Modifying a reference parameter modifies the caller's argument!

```
void swap(int& x, int& y) {
  int tmp = x;
  x = y;
  y = tmp;
}

int main(int argc, char** argv) {
  int a = 5, b = 10;

  swap(a, b);
  cout << "a: " << a << "; b: " << b << endl;
  return EXIT_SUCCESS;
}
```



Pass-By-Reference

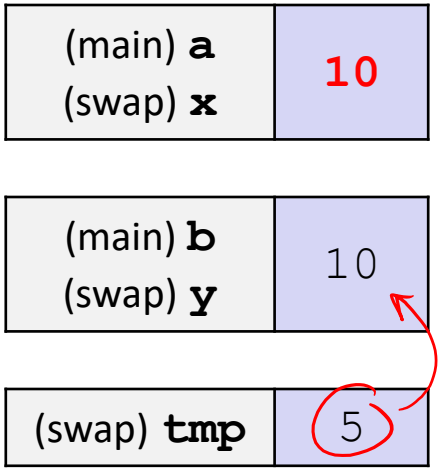
Note: Arrow points to *next* instruction.

- C++ allows you to use real pass-by-*reference*
  - Client passes in an argument with normal syntax
    - Function uses reference parameters with normal syntax
    - Modifying a reference parameter modifies the caller's argument!

```
void swap(int& x, int& y) {
  int tmp = x;
  x = y;
  y = tmp;
}

int main(int argc, char** argv) {
  int a = 5, b = 10;

  swap(a, b);
  cout << "a: " << a << "; b: " << b << endl;
  return EXIT_SUCCESS;
}
```



Pass-By-Reference

Note: Arrow points to *next* instruction.

- C++ allows you to use real pass-by-*reference*
  - Client passes in an argument with normal syntax
    - Function uses reference parameters with normal syntax
    - Modifying a reference parameter modifies the caller's argument!

```
void swap(int& x, int& y) {
  int tmp = x;
  x = y;
  y = tmp;
}

int main(int argc, char** argv) {
  int a = 5, b = 10;

  swap(a, b);
  cout << "a: " << a << "; b: " << b << endl;
  return EXIT_SUCCESS;
}
```

(main) **a** (swap) **x**	10

(main) **b** (swap) **y**	**5**

(swap) **tmp**	5

Pass-By-Reference

Note: Arrow points to *next* instruction.

- C++ allows you to use real pass-by-*reference*
  - Client passes in an argument with normal syntax
    - Function uses reference parameters with normal syntax
    - Modifying a reference parameter modifies the caller's argument!

```
void swap(int& x, int& y) {
  int tmp = x;
  x = y;
  y = tmp;
}

int main(int argc, char** argv) {
  int a = 5, b = 10;

  swap(a, b);
  cout << "a: " << a << "; b: " << b << endl;
  return EXIT_SUCCESS;
}
```

(main) **a**	10

(main) **b**	5

Lecture Outline

What else is going on?

- C++ Seems so cool!!!! What else is going on? ☺

- NSA: 1 year ago (Nov 10th, 2022)

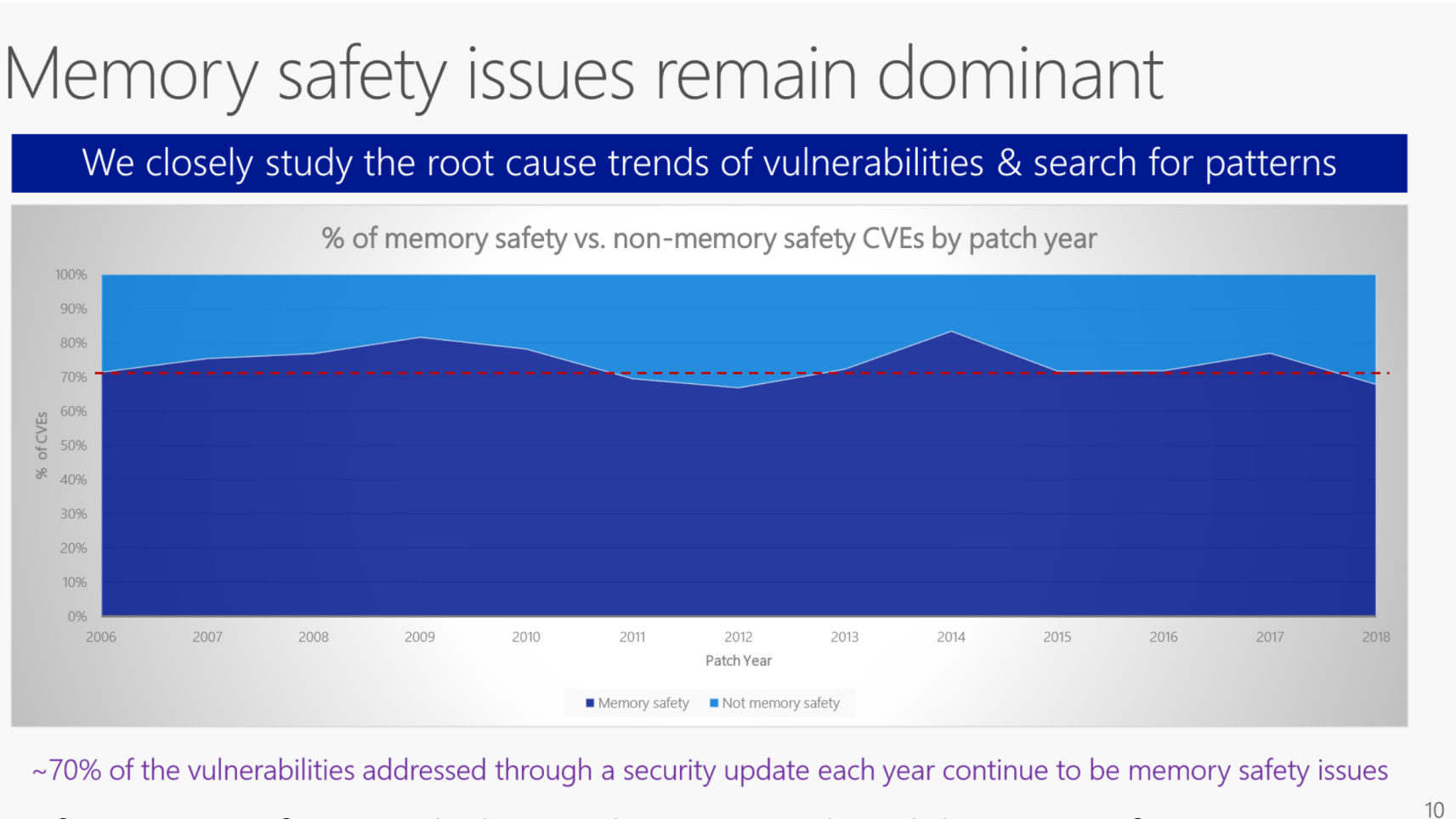
The path forward

Memory issues in software comprise a large portion of the exploitable vulnerabilities in existence. NSA advises organizations to consider making a strategic shift from programming languages that provide little or no inherent memory protection, such as C/C++, to a memory safe language when possible. Some examples of memory safe languages are C#, Go, Java, Ruby™, and Swift®. Memory safe languages provide

Rust is not mentioned in this snippet, but mentioned somewhere else in the announcement

Memory Safety CVE

- CVE = Common Vulnerabilities and Exposures



This is from Microsoft research showing how most vulnerabilities come from memory issues

Memory Safety

- Memory Safety is dominating discussion on Systems programming languages (C, C++, Rust, Zig, Nim, D, …)
- What is memory safety?
- Broadly two types:
  - Temporal Safety: making sure you don't access "objects" that are destroyed, or invalid "object" states
  - Spatial Safety: making sure you do not access memory you either shouldn't access or accessing them in the wrong ways

Temporal Safety C Example

- Here is an example in C where is the issue?

```c
int main(int argc, char** argv) {
  int* ptr = malloc(sizeof(int));
  assert(ptr != NULL);
  *ptr = 5;

  // do stuff with ptr

  free(ptr);

  printf("%d\n", *ptr);
}
```

Temporal Safety

- Here is an example in C++ where is the issue?

```
#include <iostream>
#include <vector>

using namespace std;


int main(int argc, char** argv) {
  vector<int> v {3, 4, 5};
  int& first = v.front();

  cout << first << endl;

  v.push_back(6);

  cout << v.size() << endl;
  cout << first << endl;
}
```

Temporal Safety

- Here is an example in C++ where is the issue?

```
#include <iostream>
#include <vector>

using namespace std;


void func(vector<int>& v1, vector<int>& v2) {
  v1.push_back(v2.front());
}

int main() {
  vector<int> x{3, 4, 5};
  func(x, x);
}
```

Temporal Safety

- Here is an example in C++ where is the issue?

```
#include <iostream>
#include <vector>

using namespace std;


void func(vector<int>& v1, vector<int>& v2) {
  v1.push_back(v2.front());
}

int main() {
  vector<int> x{3, 4, 5};
  func(x, x);
}
```

push_back takes in an int&

push_back may need to resize, if it does, the reference to its front becomes invalid

Spatial Safety

- C (and C++) enforce types on variables, they are statically typed
- C and C++ can easily get around the type system though:

```
int main() {
  int x = 3;
  float f1 = x;  // converts bits to floating point rep
  float f2 = *(float*)&x; // copies bits

  printf("%f\n", f1); // these two print
  printf("%f\n", f2); // different things
}
```

Spatial Safety

- C (and C++) enforce types on variables, they are statically typed
- C and C++ can easily get around the type system though:

```
int main() {
  string s = "Howdy :)";
  vector<int> v = *reinterpret_cast<vector<int>*>(&s);

  v.push_back(3);

  // this code probably crashes before getting here
}
```

Aside: unions

- A union is a type that can have more than one possible representations in the same memory position

```
union {
  float f;
  int i;
};

f = 3.14;  // assigns a float value to the union

printf("%d\n", i);  // try to interpret the same memory as an int

// this is not type checked ☹
```

Spatial Safety

- A union is a type that can have more than one possible representations in the same memory position

```
// common design pattern, return a struct that either holds
// an error or the expected value, with a bool to indicate
struct parer_result {
  bool is_valid;
  union {
    char* error message;
    struct parsed_command* cmd;
  };
};
struct parser_result parse_cmd(const char* input);

int main() {
  struct parser_result = parse_cmd("…");
  struct parsed_command = *(parser_result.cmd)
}
// We didn't check if the result was valid, may be violating
spatial safety
```

Spatial Safety

- Sometimes violating spatial safety is "needed"
  - To support “Generics” in c, we often cast to/from `void*`
  - Can be used for some cool stuff like this fast inverse square root algorithm (don’t do this, it is not fast anymore):

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y  = number;
    i  = * ( long * ) &y;                       // evil floating point bit level hacking
    i  = 0x5f3759df - ( i >> 1 );               // what the f?
    y  = * ( float * ) &i;
    y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
//  y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed

    return y;
}
```

Spatial Safety

- Spatial safety includes index out of bounds.

```
int primes[6] = {2, 3, 5, 6, 11, 13};
primes[3] = 7;
primes[100] = 0;   // memory smash!
```

No IndexOutOfBounds
Hope for segfault

- What is wrong here?

```
write(STDERR_FILENO, "Hello!\n", PAGE_SIZE);
```

- Here?

```
char buf[6];
strcpy(buf, "Hello!\n");
```

Has C++ Been Fixing These?

- C++ has been giving replacements for these features that are safer.
  - Instead of union, C++ has optional, variant, any and others
  - Instead of C arrays, there is the vector and array type

- Is this C++ safe?

```
vector<int> v {2, 3, 5, 6, 11, 13};
v[1000] = 7;        // is this safe?
v.at(1000) = 0;     // above: no, this: yes
```

- C++ Keeps adding new features that are better and safer but adding in unchecked-unsafe ways to use them. Usually, the argument is for performance

C++ Backwards compatibly

- Even with Modern C++ adding new features to get better and safer, many people stick to bad habits that are kept in C++ for backwards compatibility

Lecture Outline

- What's Next?

C++ Successor Languages

- Because of the issue with safety, 2022 has been called "the year of the C++ successor Languages"

- Just in 2022, three successor languages were announced:
  - Val (now called Hylo)
  - Carbon
  - cppfront (sometimes called cpp2)

- There have been many languages before:
  - D
  - Go
  - Rust
  - Others: Nim, Zig, Swift, etc.

C and C++ are used everywhere

- Many things are written largely/primarily in C++ or C
  - The Adobe suite (Photoshop, etc)
  - The Microsoft office suite (word, PowerPoint, etc.)
  - The libre office suite (FOSS word, PowerPoint, etc)
  - Chromium (Core of most web browsers, Edge, Opera, Chrome, etc)
  - Firefox
  - Most Database implementations
  - Tensorflow & Pytorch
  - gcc, clang & llvm (which is the backbone for many compilers)
  - Game Engines (Unreal, Unity, etc.)

Most of this information is from Jason Turner's "C++ is 40… Is C++ DYING?" video https://www.youtube.com/watch?v=hxjSpasg3gk

C and C++ are used everywhere

- Regularly ranks in top used ~5-10 programming languages

- Many people still use C++
  - Estimates from JetBrains
  - ~1,157,000 professional developers use C++ as their primary language
  - ~2,492,000 professional developers regularly use C++

Programming Language Adoption

I do believe that there is real value in pursuing functional programming, but ***it would be irresponsible to exhort everyone to abandon their C++ compilers*** *and start coding in Lisp, Haskell, or, to be blunt, any other fringe language.*

To the eternal chagrin of language designers, there are plenty of externalities that can overwhelm the benefits of a language…

We have ***cross platform*** *issues, proprietary* ***tool chains****,* ***certification*** *gates,* ***licensed*** *technologies, and stringent* ***performance*** *requirements on top of the issues with* ***legacy*** *codebases and* ***workforce*** *availability that everyone faces. …*

— John Carmack [***emphasis*** *added]*

45

For better or for worse, C++ already exists and has a bunch of work behind it.
Moving to another thing is going to take time and money, but is not impossible

Screenshot from Herb Sutter's Plenary in cppcon 2023: https://www.youtube.com/watch?v=8U3hl8XMm8c
It is an interesting talk, but his cppcon 2022 or c++now 2023 talks may be better starting points for those interested

Migration

- Some organizations are (at least in part) trying to move from C / C++
- The Linux kernel has incorporated Rust into it
  - It never allowed C++ into the kernel
- Microsoft and Mozilla Firefox are putting in a lot of effort to start training some employees to program in Rust.
- The situation is developing, we will see how things evolve over time ☺

bleg

Future Courses

- Systems Courses
  - CIS 3410: Compilers
  - CIS 5050: Software Systems
  - CIS 5530: Networked Systems
  - CIS 5550: Internet and Web Systems
  - CIS 5500: Database and Information Systems
  - CIS 5470: Software Analysis
- Otherwise related courses
  - CIS 5600 Interactive Computer Graphics
  - CIS 5610 Advanced Computer Graphics
  - CIS 5650 GPU Programming and Architecture
  - CIS 3310 Security
  - CIS 5510 (Also security, may have remembered the # wrong)

Thanks for a great semester!

- Special thanks to all the instructors before me (Both at UPenn and UW) who have influenced me to make the course what it is

- Huge thanks to the course TA's for helping with the course!

Head Teaching Assistants

Nate Hoaglund
He/Him

nhoag@seas

Seungmin Han
he/him/his

hanseun@seas

Teaching Assistants

Andy Jiang
he/him/his

jianga@wharton

Audrey Yang
she/her

auyang@seas

August Fu
he/him

yuntongf@seas

Daniel Da
he/him

dadaniel@wharton

Ernest Ng
he/him/his

ngernest@seas

Haoyun Qin
He/him/his

qhy@seas

Jason Hom
He/Him/His

jasonhom@seas

Jeff Yang
he/his/him

yjeffrey@seas

Jerry Wang
he/his/him

jwang01@seas

Jinghao Zhang
She/her/hers

jinghaoz@seas

Kevin Bernat
He/Him/His

kbernat@seas

Leon Hertzberg
He/Him

leonjh@seas

Maxi Liu
she/her

maxiliu@seas

Ria Sharma
she/her

rias@wharton

Rohan Verma
He/Him/His

rover@seas

Ryoma Harris
he/him

ryomah@seas

Shyam
he/him

smehta1@seas

Tina Kokoshvili
she/her/hers

tinatin@seas

Zhiyan Lu
She/her/hers

zhiyanlu@sas

Thanks for a great semester!

- Thanks to you!
  - It has been another tough semester. Still not completely out of the pandemic, Zoom fatigue, faltering motivation, etc
  - My First offering of the course, things are still a bit rough
  - You've made it through so far, be proud that you've made it and what you've accomplished!

- **Please take care of yourselves, your friends, and your community**

Ask Me Anything