Week 1: Lecture B
Python, Debugging, and VM Setup

Thursday, August 24, 2023
Reminders

- Be sure to join the course **Canvas, Gradescope, and Piazza**
  - See links at top of course page
  - [http://cs4440.eng.utah.edu](http://cs4440.eng.utah.edu)

- Trouble accessing? See me after class!
  - Or email me at: snagy@cs.utah.edu
Reminders

**Course website** .................... wiki, assignments, schedule, slides, office hours

**Piazza** ......................................................... questions, discussion, announcements

**Gradescope** .......................................................... weekly post-lecture quizzes

**Canvas** ........................................................ project submission, course gradebook

**Instructor email** ([snagy@cs.utah.edu](mailto:snagy@cs.utah.edu)) .................. administrative issues
Reminders

- First weekly **Lecture Quiz** released on **Gradescope**
  - Submit by **11:59PM this Monday**
  - Late submissions not accepted

- Lecture quizzes released after Tuesday’s lecture
  - Due the following Monday
  - Covers content from both Tuesday + Thursday lectures
Reminders

- **Poll Everywhere:** check your UMail for an **account registration** email
  - We won’t count today’s attendance, but be sure to sign up before next week!

- **Use your** [UID@utah.edu](mailto:UID@utah.edu) **when participating**
  - Should work automatically if you got the sign-up email
  - **Test this out today**—if it does not work, please see me!
Reminders

- **Officers Hours**
  - [http://cs4440.eng.utah.edu](http://cs4440.eng.utah.edu)
  - Cancellations announced via Piazza
  - Busier near deadlines—**start early!**
Reminders

- Can work in **teams of up to two**
  - Find teammates on **Piazza**
  - Post on

- Why work with someone else?
  - Pair programming
  - Divide and conquer
  - Two sets of eyes to solve problems
  - Teaching others helps you learn more

- Yes, you are free to work solo...
  - But we encourage you to team up!
Announcements

- **Project 1: Crypto** releasing on **Tuesday, August 29**
  - **Deadline:** Thursday, September 21st by 11:59PM
Announcements

First meeting **TONIGHT @ 5PM**
in MEB 3147
Questions?
Last time on CS 4440...

The Security Mindset
Modeling the Attacker
Assessing Risk
Secure Design
“Computer security studies how systems behave in the presence of an adversary.”

- The adversary...
- a.k.a. the attacker
- a.k.a. the bad guy
- An intelligence that actively tries to cause the system to misbehave.
The Security Mindset

- **Thinking like a defender**
  - Know what you’re defending, and against whom
  - Weigh benefits vs. costs: No system is ever completely secure.
  - Embrace “rational paranoia”

- **Thinking like an attacker**
  - Understand techniques for circumventing security
  - Look for ways security can break, not reasons why it won’t
Thinking as a Defender

- Security policy
  - What are we trying to protect?
  - What properties are we trying to enforce?
- Threat model
  - Who are the attackers? Capabilities? Motivations?
  - What kind of attack are we trying to prevent?
- Risk assessment
  - What are the weaknesses of the system?
  - What will successful attacks cost us?
- How likely?
  - Countermeasures
  - Costs vs. benefits?
  - Technical vs. nontechnical?

The challenge is to think rationally and rigorously about risk. Rational paranoia.
Threat Models

- Who are our adversaries?
  - Motives?
  - Capabilities?
  - Level of access?

- What kinds of attacks must we prevent?
  - Think like the attacker!

- Limits: kinds of attacks we should ignore?
  - Unrealistic versus unlikely
Security through... obscurity?

Common mistake:
- Trying to convince yourself the system is secure since attacker won't know $X$

Better approach:
- Limit the assumptions that the security of your system depends upon
- Assume the attacker knows everything but a *small* bit of data (e.g., a key)
Assessing Risk

- Remember: *Rational* paranoia

- What would security breaches cost us?
  - Direct: money, intellectual property, safety
  - Indirect: reputation, future business, well being

- How likely are these costs?
  - Probability of attacks?
  - Probability of success?
Countermeasures

- Technical countermeasures
  - Bug fixes, more crypto, re-architecting, etc.

- Nontechnical countermeasures
  - Law, policy (government, institutional)
  - Procedures, training, auditing, incentives, etc.
Costs of Security

- **No security mechanism is free**

- **Direct costs:**
  - Design, implementation, enforcement, false positives

- **Indirect costs:**
  - Lost productivity, added complexity, time to market

- **Challenge is to rationally weigh costs vs. risk**
  - Human psychology makes reasoning about high cost, low probability events very difficult
Exercises

Should you use a strong password online?

- Assets?
- Adversaries?
- Risk assessment?
- Countermeasures?
- Costs/benefits?
Exercises

Using a credit card safely?

- Assets?
- Adversaries?
- Risk assessment?
- Countermeasures?
- Costs/benefits?
High-level Approaches

Attacks

Defenses
Questions?
This time on CS 4440...

Intro to Python
Debugging Code
Course VM Setup
Projects cover a few languages and tools:

- **Project1**: Python 3
- **Project2**: C/C++, x86, GDB
- **Project3**: SQL, HTML, JavaScript
- **Project4**: Python 3, Wireshark
Languages and Tools in CS 4440

- Projects cover a few languages and tools:
  - **Project 1**: Python 3
  - **Project 2**: C/C++, x86, GDB
  - **Project 3**: SQL, HTML, JavaScript
  - **Project 4**: Python 3, Wireshark

- This may seem daunting—but don’t panic!
Languages and Tools in CS 4440

Projects cover a few languages and tools:

- **Project 1**: Python 3
- **Project 2**: C/C++, x86, GDB
- **Project 3**: SQL, HTML, JavaScript
- **Project 4**: Python 3, Wireshark

This may seem daunting—but don’t panic!

- Only using a **small subset** of their capabilities
- We’ll cover some basics in lecture as we go along
- We’ll post resources for you on the **CS 4440 Wiki**
Have you browsed CS 4440 Wiki yet?

Yes! 0%
No :( 0%
An Intro to Python 3
Python 3

- Primary language for your Projects
  - Though expect to see some others too

- Characteristics:
  - High-level
  - Interpreted
  - Object Oriented
  - Dynamically Typed
  - Lots of indentation

```python
print("Hello, world!")
```
Running Python Code

- Interactive mode
  - Launch Python 3 console
  - Enter code line-by-line
  - Executed line-by-line

```bash
$ python3
>>> print("Hello from the interpreter!")
Hello from the interpreter!
>>> exit()
```
### Running Python Code

- **Scripting mode**
  - Edit your script (e.g., `MyScript.py`)
  - Then call the `python3` binary on it

```bash
$ cat MyScript.py
#!/usr/bin/python3
print("Hello from scripting mode!")
$ python3 MyScript.py
Hello from scripting mode!
```
You’ll be writing relatively simple scripts
- No need for an IDE
- IDEs can/will break things

Recommended text editors:
- VIM
- Nano
- Emacs
- FeatherPad
- Many others—pick one you like!
Variables

- Can contain alphanumerical characters and some special characters

- Common conventions:
  - Variable names that start with lower-case letters
  - Class names beginning with a capital letter

- Some keywords are **reserved** (cannot be used as variable names)
  - Examples: `and`, `continue`, `break`
  - Python will complain if you use these

- **Dynamically typed**: a variable’s type is derived from its value
Variables

Types you’ll likely see:
- Integer (*int*)
- Float (*float*)
- String (*str*)
- Boolean (*bool*)
- Custom classes (e.g., *md5*)
Variables

- **Types you’ll likely see:**
  - Integer (int)
  - Float (float)
  - String (str)
  - Boolean (bool)
  - Custom classes (e.g., md5)

- **Variable assignment:**
  - Assignment uses the “=” sign

```python
>>> x = 5
>>> print(type(x))
<class 'int'>
```
Variables

- Types you’ll likely see:
  - Integer (`int`)
  - Float (`float`)
  - String (`str`)
  - Boolean (`bool`)
  - Custom classes (e.g., `md5`)

- Variable assignment:
  - Assignment uses the “=” sign
  - Value changed? **So does type!**

```python
>>> x = 5
>>> print(type(x))
<class 'int'>

>>> x = "cs4440"
>>> print(type(x))
<class 'str'>
```
Variables

- **Casting:**
  - Pick a desired data type
  - “Wrap” your variable in it

```python
>>> x = 5
>>> print(x, type(x))
5 <class 'int'>
```
Casting:
- Pick a desired data type
- “Wrap” your variable in it
- **Re-casting** will change type!

```
>>> x = 5
>>> print(x, type(x))
5 <class 'int'>

>>> x = float(x)
>>> print(x, type(x))
5.0 <class float>
```
You will use **strings** in many exercises

- Super flexible to use and manipulate
- We’ll cover some basic conventions

```python
>>> x = "odoyle"
```
You will use **strings** in many exercises
  - Super flexible to use and manipulate
  - We’ll cover some basic conventions

Basic string manipulation:
  - Length

```python
>>> x = "odoyle"
>>> print(len(x))
6
```
You will use **strings** in many exercises
- Super flexible to use and manipulate
- We’ll cover some basic conventions

**Basic string manipulation:**
- Length
- Appending

```python
>>> x = "odoyle"
>>> print(len(x))
6

>>> print(x + "rules")
odoylerules
```
You will use **strings** in many exercises
- Super flexible to use and manipulate
- We’ll cover some basic conventions

**Basic string manipulation:**
- Length
- Appending
- Substrings

```python
>>> x = "odoyle"
>>> print(len(x))
6

>>> print(x + "rules")
odoylerules

>>> print("odoy" in x)
True
```
- Other string manipulations:

```python
>>> x = "cs4440:fa23"
```
Strings

- Other string manipulations:
  - Splitting by a delimiter

```python
>>> x = "cs4440:fa23"
>>> print(x.split(':'))
['cs4440', 'fa23']
```
Other string manipulations:
- Splitting by a delimiter
- Stripping characters

```python
>>> x = "cs4440:fa23"
>>> print(x.split(':'))
['cs4440', 'fa23']

>>> print(x.strip(':'))
cs4440fa23
```
### Strings

- **Other string manipulations:**
  - Splitting by a delimiter
  - Stripping characters
  - Repeating characters

```
>>> x = "cs4440:fa23"
>>> print(x.split(':'))
['cs4440', 'fa23']

>>> print(x.strip(':'))
cs4440fa23

>>> print('A'*10)
AAAAAAAAAAAA
```
Sometimes you will work with data as **bytes**
- In Python, **byte strings** appear as `b'data'`

**Examples:**
- **Encoding** to a byte string

```python
>>> x = "cs4440"
>>> x = x.encode('utf-8'))
>>> print(x, type(x))
b'cs4440' <class 'bytes'>
```
Sometimes you will work with data as **bytes**

- In Python, **byte strings** appear as `b'data'`

**Examples:**

- **Encoding** to a byte string
- **Decoding** a byte string

```python
>>> x = "cs4440"
>>> x = x.encode('utf-8'))
>>> print(x, type(x))
b'cs4440' <class 'bytes'>

>>> y = x.decode('utf-8'))
>>> print(y, type(y))
cs4440 <class 'str'>
```
Sometimes you will work with data as **bytes**
- In Python, **byte strings** appear as `b'data'

**Examples:**
- **Encoding** to a byte string
- **Decoding** a byte string
- Must keep the same codec (e.g., `utf-8`)

```python
>>> x = "cs4440"
>>> x = x.encode('utf-8'))
>>> print(x, type(x))
b'cs4440' <class 'bytes'>

>>> y = x.decode('utf-8'))
>>> print(y, type(y))
cs4440 <class 'str'>
```
Sometimes you will work with data as **bytes**
- In Python, *byte strings* appear as `b'data'`

**Examples:**
- **Encoding** to a byte string
- **Decoding** a byte string
  - Must keep the same codec (e.g., utf-8)

**Conceptually can be a little confusing**
- Functions `print()` and `type()` are your friends!

```python
>>> x = "cs4440"
>>> x = x.encode('utf-8'))
>>> print(x, type(x))
b'cs4440' <class 'bytes'>

>>> y = x.decode('utf-8'))
>>> print(y, type(y))
cs4440 <class 'str'>
```
Other Key Concepts

- A few other concepts to review
  - Check these out in the CS 4440 Wiki

CS 4440 Wiki: All Things CS 4440

This Wiki is here to help you with all things CS 4440: from setting up your VM to introducing the languages and tools that you’ll use. Check back here throughout the semester for future updates.

Have ideas for other pages? Let us know on Piazza!

Tutorials and Cheat Sheets

<table>
<thead>
<tr>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM Setup &amp; Troubleshooting</td>
<td>Instructions for setting up your CS 4440 Virtual Machine (VM).</td>
</tr>
<tr>
<td>Terminal Cheat Sheet</td>
<td>Navigating the terminal, manipulating files, and other helpful tricks.</td>
</tr>
<tr>
<td>Python 3 Cheat Sheet</td>
<td>A gentle introduction to Python 3 programming.</td>
</tr>
<tr>
<td>GDB Cheat Sheet</td>
<td>A quick reference for useful GNU Debugger (GDB) commands.</td>
</tr>
<tr>
<td>JavaScript Cheat Sheet</td>
<td>A gentle introduction to relevant JavaScript commands.</td>
</tr>
</tbody>
</table>
Other Key Concepts

- A few other concepts to review
  - Check these out in the **CS 4440 Wiki**

- Lists
  - Appending
  - Prepending
  - Insert, Remove

### List Manipulation

#### Indexing:

```python
>>> x = ['cs4440', 'is', 'cool']
>>> print(x[0])
cs4440

>>> x = ['cs4440', 'is', 'cool']
>>> print(x[-1])
cool
```

#### Inserting:

```python
>>> x = ['cs4440', 'is', 'cool']
>>> x[-1] = 'fun'
>>> print(x)
['cs4440', 'is', 'fun']

>>> x.insert(2, 'super')
>>> print(x)
['cs4440', 'is', 'super', 'fun']
```

#### Joining:

```python
>>> x = ['cs4440', 'is', 'cool']
>>> print(' '.join(x))
cs4440 is cool

>>> y = ['all', 'day']
>>> print(x + y)
['cs4440', 'is', 'super', 'cool', 'all', 'day']
```
Other Key Concepts

- A few other concepts to review
  - Check these out in the **CS 4440 Wiki**

- Lists
  - Appending
  - Prepending
  - Insert, Remove

- Control Flow
  - Loops
  - If/Else Statements

---

**List Manipulation**

**Indexing:**

```python
>>> x = ['cs4440', 'is', 'cool']
>>> print(x[0])
cs4440
```

**Appending:**

```python
>>> x = ['cs4440', 'is', 'cool']
>>> append(x, 'fun')
```

**Prepending:**

```python
>>> x = ['cs4440', 'is', 'cool']
>>> prepend(x, 'super')
```

**Insert, Remove:**

```python
>>> x = ['cs4440', 'is', 'cool']
>>> print(x[0])
```

---

**Conditional Statements**

**If statements:**

```python
>>> x = 5
>>> if (x & 2 == 1):
...     print("Yes")
# Prints string 'Yes' if condition is True.
```

**Else statements:**

```python
>>> x = 5
>>> if (x % 3 == 1):
...     print("Yes")
... else:
...     print("Nope")
# Prints 'Nope!' if the condition is False.
```

**Loops**

**For loops:**

```python
>>> x = ['a', 'b', 'c']
>>> for y in x:
...     print(y)
0
b
```

**While loops:**

```python
>>> x = 3
>>> while x != 0:
...     x -= 1
3
2
1
```
Other Key Concepts

- A few other concepts to review
  - Check these out in the **CS 4440 Wiki**

- Lists
  - Appending
  - Prepending
  - Insert, Remove

- Control Flow
  - Loops
  - If/Else Statements

- Functions
Questions?
Debugging Your Code
What will the following code do?

```python
age = input("How old are you? ")
next_age = age + 1
print("Next year you will be", next_age)
```
What will the aforementioned code do?

<table>
<thead>
<tr>
<th>Option</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Print a number (your age + 1)</td>
<td>0%</td>
</tr>
<tr>
<td>Print a string (your age + 1)</td>
<td>0%</td>
</tr>
<tr>
<td>All of the above!</td>
<td>0%</td>
</tr>
<tr>
<td>None of the above</td>
<td>0%</td>
</tr>
</tbody>
</table>
What will the following code do?

```python
age = input("How old are you? 
next_age = age + 1
print("Next year you will be", next_age)
```
Where to begin debugging?

- Errors say where the error is!
  - Filename
  - Line number
  - The actual line of code

```
Traceback (most recent call last):
  File "MyScript.py", line 2, in <module>
    next_age = age + 1
TypeError: must be str, not int
```
Where to begin debugging?

- Errors say where the error is!
  - Filename
  - Line number
  - The actual line of code

```python
Traceback (most recent call last):
 File "MyScript.py", line 2, in <module>
   next_age = age + 1  
TypeError: must be str, not int
```
Where to begin debugging?

- Errors say where the error is!
  - Filename
  - Line number
  - The actual line of code

- The error’s root cause:
  - Program tried "29" +1
  - Strings and numbers are different data types!

Traceback (most recent call last):
File "MyScript.py", line 2, in <module>
  next_age = age + 1
TypeError: must be str, not int
Where to begin debugging?

- Errors say where the error is!
  - Filename
  - Line number
  - The actual line of code

- The error’s root cause:
  - Program tried "29" + 1
  - Strings and numbers are different data types!

- **The fix:** cast age as an int

```python
age = input("How old are you? ")
next_age = int(age) + 1
```
Debugging is a Process

- **Remember:** `print()` and `type()` are your friend!
  - Insert these, re-run your program, and check output
  - Does the output match what you expect?
  - If not, investigate further and try again!
Debugging is a Process

- **Remember**: `print()` and `type()` are your friend!
  - Insert these, re-run your program, and check output
  - Does the output match what you expect?
  - If not, investigate further and try again!
Lazy Debugging

My code doesn't work. I don't know why! Help me!!!!
Asking for Help

- It’s perfectly fine to ask for help
  - That’s what we / Piazza are here for!
Asking for Help

- It’s perfectly fine to ask for help
  - That’s what we / Piazza are here for!

- Help others help you! Explain:
  - What error code are you getting?
  - What do you think it means?
  - What fixes have you tried?
  - What fixes did not work?
It’s perfectly fine to ask for help
- That’s what we / Piazza are here for!

Help others help you! **Explain:**
- What error code are you getting?
- What do you think it means?
- What fixes have you tried?
- What fixes did not work?

Please try to avoid “**instructor private posts**” about debugging your code
- We get a lot of these near deadlines—it becomes impossible to keep up / help everyone!
- We may un-private your post if it contains information that’s useful for the class 😊
Questions?
VM Setup
Virtual Machines (VM)

- Why do we use a VM in this course?
  - Minor software differences can totally break things
  - We want everyone to have the same environment
  - We'll grade everyone using the same environment
Virtual Machines (VM)

Why do we use a VM in this course?
- Minor software differences can totally **break things**
- We want everyone to have the **same environment**
- We’ll **grade** everyone using the same environment

Open the **CS 4440 Wiki**
- See the **VM Setup** page
- Follow the instructions
- Once your VM is setup, you are free to leave!
- In the meantime, feel free to **ask questions**

Course Homepage: [http://cs4440.eng.utah.edu](http://cs4440.eng.utah.edu)
Next time on CS 4440...

Message integrity (a.k.a. applied cryptography)