This time

• Administrativia

• Whirlwind tour

• Buffer overflows

• Trusting trust

Administrativia

Communicating

• Resources and all this info will be on the class website
  • http://triffid.cs.umd.edu/414

• Who
  • Me: Pete Keleher (keleher@cs.umd.edu)
  • TAs: Chengxi Ye, Katura Harvey, Kartik Nayak, and Lee Williams
  • Office hours are (will be) on the website
  • If my office hours don’t work for you, email and set up a time

• We will be using Piazza
  • You should have been added.
Administrative

Textbooks

- None required
  - Mostly in-class and papers posted on website

- Recommended texts, if you are so inclined
  - “Security in Computing”, Pfleeger & Pfleeger
  - “Security Engineering”, Ross Anderson
    - Free online: http://www.cl.cam.ac.uk/~rja14/book.html

Administrative

Outside reading

- The best way to learn is to reinforce

- Lots of security resources (something is always breaking).
  - Krebs on security
  - Bruce Schneier’s blog
  - Any other favorites? Let us know on Piazza
What’s in this course?

Software Security

How do we build software that is secure?

Memory safety

Attacks on TCP & DNS

Malware

Web security

Underground sp

Static analysis

Design principles

What it is, and how to use it responsibly.

A black-box approach to crypto

Designing protocols that use crypto

Anonymity: hiding who you are

How to build secure networked systems.

Authentication: proving who you are

What’s in this course?

Software Security

How do we build software that is secure?

Crypto

What it is, and how to use it responsibly.

Network Security

How to build secure networked systems.

Attacks and defenses across all of these
Ethics and legality

- You will be learning about (and implementing and launching) attacks, many of which are in active use today.

  *This is not an invitation to use them without the explicit written consent of all parties involved*

- If you want to try something out, then *let me know* and I will try to help create a safe environment

- This is not just a question of ethics; to do otherwise would risk violating UMD policies and MD/USA laws
Prerequisite knowledge

• You should be reasonably proficient in C and Unix

• You should also be creative and resourceful (those who try to attack your systems will be!)

• Otherwise, this course won’t require any prior knowledge in networking or crypto

What’re grades based on?

• Grade breakdown
  • Projects (4 x ~11% each)
  • Midterms (2 x 15% each)
  • Final (25%)
Midterms & Exams

Expected dates

Midterm #1: Oct. 7 15%
Midterm #2: Nov. 18 15%
Final exam: Dec. 16 25%

Please see the syllabus for information about excused absences

A brief whirlwind tour of some things to come
Is anything really “secure”?

- Security requires context
  - What is the **threat model**? What can the attacker do?
  - What are the **assets** you seek to protect?
  - Whom and what do you **trust**?

- “Trust no one!”
  - That’s the spirit!
  - But how did you compile your code again?
  - Who built your OS? Your hardware?…

**Required reading**

“Reflections on Trusting Trust”
Ken Thompson

Online shopping

**How do you know you’re really talking to amazon.com?**

- Transitive trust
  - You trust your hardware manufacturer / OS
  - They trust a company called Verisign
  - Verisign trusts that amazon.com is run sanely.

- This is what your browser toolbar is showing you. But what about:
  - Your browser itself?
  - Your OS?
  - The compiler that compiled all of the above?
  - The hardware manufacturer on which all of the above run?
Classes of security

Confidentiality

The concealment of information

- Some common ways to achieve confidentiality
  - Encryption (only those with the right keys can decrypt)
  - Access control: defining & enforcing “need to know”
  - Authorization: proving that a person/system is allowed access
  - Authentication: proving that you are somebody
    - By something you have (a key fob, a physical key)
    - By something you are (fingerprints, retina scans)
    - By something you know (password, a secret key)

- Variant: Anonymity (hiding who is communicating)
Integrity

Prevention of unauthorized changes

• Think “the telephone game”: how do you know it’s the same message at the end?

• Some common ways to achieve integrity
  • Backups (redundancy to recover from large failures)
  • Checksums (redundancy to detect failures)
  • Error correcting codes (redundancy to detect and fix small failures)
  • Encryption/signatures (to detect even small modifications)

• Variant: Authenticity (identification and assurance of data origin)
  • i.e., not only has the data not changed, it’s really from Amazon

Availability

Ability to use the info/resources desired

• Types of attacks
  • Denial of service (send more data than the service supports)
  • Concurrency bugs (hold locks forever)

• Some common ways to achieve availability
  • Computation (Unix ‘nice’ values)
  • Network (load balancing)
  • Storage (user quotas)

• Variant: Fairness (does everyone get a fair share of the resources, for many definitions of “fair”)

Analyzing security

Takeaway points

• Analyzing security requires a whole-systems view
  • Hardware, software, users, economics, ….

• Security is only as strong as the weakest link
  • May have been difficult to break into the building
  • But if the data is sent unencrypted…

• Securing a system can be difficult
  • Interdisciplinary (software, hardware, UI design)
  • Humans are in the loop

• Security through obscurity does not work
  • Especially for high-value assets
  • It’s only a matter of time until someone finds out

The Goals of CMSC 414

Be able to eliminate bugs and design flaws and/or make them harder to exploit.

Be able to think like attackers.

Develop a foundation for deeply understanding the systems we use and build.

Software  Hardware  Protocols
Users  Law  Economics
This time

We will begin our 1st section: **Software Security**

By investigating Buffer overflows and other memory safety vulnerabilities

• History

• Memory layouts

• Buffer overflow fundamentals

Software security

• Security is a form of dependability
  • Does the code do “what it should”
  • To this end, we follow the software lifecycle

• Distinguishing factor: an *active, malicious* attacker

• Attack model
  • The developer is trusted
  • But the attacker can provide any inputs
    - Malformed strings
    - Malformed packets
    - etc.

What harm could an attacker possibly cause?
screensaver --prompt=“Don’t unlock plz”

Don't unlock plz
Locked by pete
press ctrl-c to logout

screensaver --prompt=“Don’t unlock pretty plz”

Don't unlock pretty plz
Locked by pete
press ctrl-c to logout
screensaver --prompt="Don’t unlock plz
Locked by pete
press ctrl-c to logout"

Don't unlock plz

Locked by pete
press ctrl-c to logout

screensaver -prompt="Under maintenance;
Do not interrupt"

Under maintenance;
Do not interrupt

Locked by pete
press ctrl-c to logout
We’re going to focus on C

C is still very popular

Many mission critical systems are written in C

• Most kernels & OS utilities
  • fingerd
  • X windows server

• Many high-performance servers
  • Microsoft IIS
  • Microsoft SQL server

• Many embedded systems
  • Mars rover

• But the techniques apply more broadly
  • Wiibrew: “Twilight Hack” exploits buffer overflow when saving the name of Link’s horse, Epona

http://www.tiobe.com
We’re going to focus on C

The harm can be substantial

• 1988
  • Morris worm
    • Propagated across machines (too aggressively, thanks to a bug)
    • One way it propagated was a buffer overflow attack against a vulnerable version of fingerd on VAXes
      • Sent a special string to the finger daemon, which caused it to execute code that created a new worm copy
      • Didn’t check OS: caused Suns running BSD to crash
    • End result: $10-100M in damages, probation, community service

  Robert Morris is now a professor at MIT

• 1988
  • CodeRed
    • Exploited an overflow in the MS-IIS server
    • 300,000 machines infected in 14 hours
We’re going to focus on C

The harm can be substantial


- SQL Slammer
  - Exploited an overflow in the MS-SQL server
  - 75,000 machines infected in 10 minutes

**GHOST**: glibc vulnerability introduced in 2000, only just announced six months ago
Buffer overflows are prevalent

Percent of all vulnerabilities


Buffer overflows are prevalent

Total number of buffer overflow vulnerabilities

Our goals

- Understand how these attacks work, and how to defend against them

- These require knowledge about:
  - The compiler
  - The OS
  - The architecture

Analyzing security requires a whole-systems view
Memory layout

Refresher

• How is program data laid out in memory?

• What does the stack look like?

• What effect does calling (and returning from) a function have on memory?

• We are focusing on the Linux process model
  • Similar to other operating systems
All programs are stored in memory

The process’s view of memory is that it owns all of it

In reality, these are virtual addresses; the OS/CPU map them to physical addresses

The instructions themselves are in memory

...
Data’s location depends on how it’s created

We are going to focus on runtime attacks

Stack and heap grow in opposite directions

Compiler provides instructions that adjusts the size of the stack at runtime

apportioned by the OS; managed in-process by malloc

Focusing on the stack for now