Computer Security

ENEE 657

Prof. Tudor Dumitraș
Assistant Professor, ECE
University of Maryland, College Park

About Me

Tudor Dumitraș

Office: AVW 3425

Email: `tdumitra@umiacs.umd.edu`

Course Website: http://ter.ps/enee657

My Background

- Ph.D. at Carnegie Mellon University
  - Research in distributed systems and fault-tolerant middleware
- Worked at Symantec Research Labs
  - Built WINE platform for security analytics
    - Used for sharing real-world security telemetry with academic researchers
    - One of the early 'threat intelligence' platforms
- Joined UMD faculty
- Data-driven security (measurements, machine learning, malware)
  - Focus on solving security problems with data analysis techniques

WINE

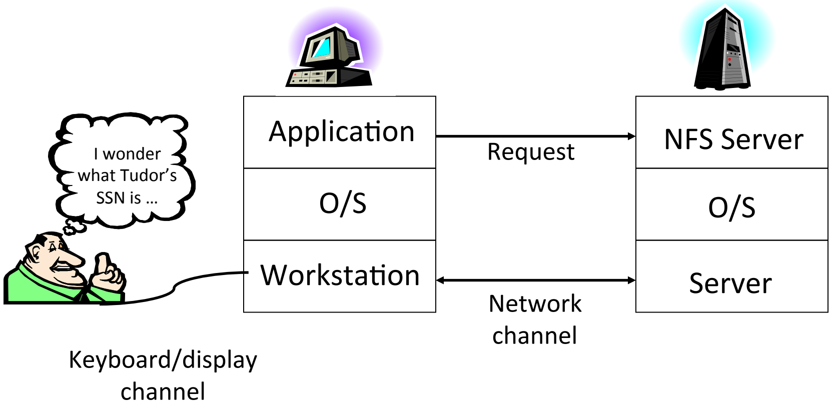
3

ENEE 657 in a Nutshell

- ENEE 657 is a **graduate**-level security course
  - Learn by **reading**, **explaining** and **doing**
  - **Project oriented**: develop to a degree that would merit **publication** in one of the **workshops** associated with the USENIX Security Symposium 2018
- Aims to prepare you for **research in security**
  - **Not** a tutorial or comprehensive course on these topics
  - Instead, exploring a range of topics to illustrate some of the current research challenges
  - Targeted at students who want to conduct research in the area or who are more generally interested in security or distributed systems

4

Who Can You Trust?



I wonder what Tudor's SSN is …

Application

O/S

Workstation

Request

NFS Server

O/S

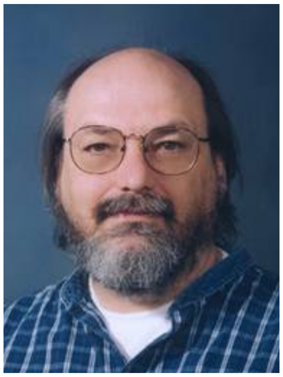
Server

Network channel

Keyboard/display channel

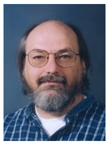
- Where is the request "from"?
  - The user? The workstation? The application? The network channel? All of the above?
  - Which of these actors do you trust?

Ken Thompson



ACM Turing Award, 1983

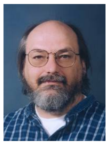
"Reflections on Trusting Trust"



- What software can we trust?
- Example: any operating system includes a program checking whether users are allowed to log in
  - "login" or "su" in Unix
  - Is the login binary from Windows/Mac OS/Ubuntu/etc. trustworthy?
  - Does it send your password to someone?
  - Does it have backdoor for a "special" remote user?
- Can't trust the binary, so check source code or write your own, recompile
- Does this solve problem?

7

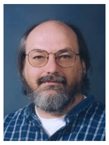
"Reflections on Trusting Trust" – cont'd



- Who wrote the compiler?
- Compiler looks for source code that looks like the login process, inserts backdoor into it
- Ok, inspect the source code of the compiler… Looks good? Recompile the compiler!
- Does this solve the problem?

8

"Reflections on Trusting Trust" – cont'd



- The UNIX login program is compiled by a C compiler
  - The C compiler was also compiled by an (older) C compiler
- Aside: how does the compiler handle special characters?

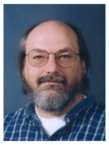
```
…
c = next( );
if(c != '\\')
        return(c);
c = next( );
if(c == '\\')
        return('\\');
if(c== 'n')
        return('\n');
if(c == 'v')
        return(11);
…
```

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

In future versions of the compiler: use the special character

When adding a new special character to the C language, must specify the character code

"Reflections on Trusting Trust" – cont'd

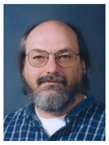


- The compiler is written in C …

```
compiler(S) {
      if (match(S, "login-pattern")) {
            compile (login-backdoor)
            return
      }
      if (match(S, "compiler-pattern")) {
            compile (compiler-backdoor)
            return
      }
      .... /* compile as usual */
   }
```

In future versions of the compiler: the backdoor no longer appears in the source code

"Reflections on Trusting Trust" – cont'd



"The moral is obvious. You can't trust code that you did not totally create yourself. (Especially code from companies that employ people like me.)"

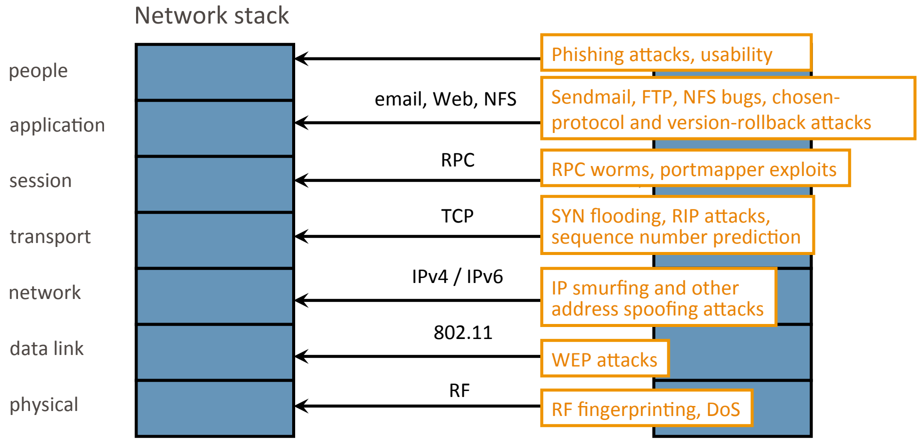
What Can Attackers Do?

- **Attack targets**: clients, servers, networks, applications, users
- Example **attack methods**:
  - **End-hosts (or devices)**: install malware
  - **LAN**: read, replay, insert, delete, block messages
  - **Internet**: send spam, conduct distributed denial of service attacks
  - **Applications**: exploit vulnerabilities
  - **Data**: steal/corrupt secret data, plant invalid data
  - **Users**: conduct social engineering attacks

Aside: Is Hardware Secure?

- Malicious device firmware
  - Some HW functionality is actually implemented in SW
  - Do you trust device firmware to come from legitimate vendor?
  - Is firmware free of vulnerabilities?
- Malicious hardware
  - HW is as complex as SW and is designed using SW tools
  - Do you know where each HW component comes from?
  - Can you authenticate your HW?
  - Could the CAD tools have introduced a backdoor (HW trojan)?

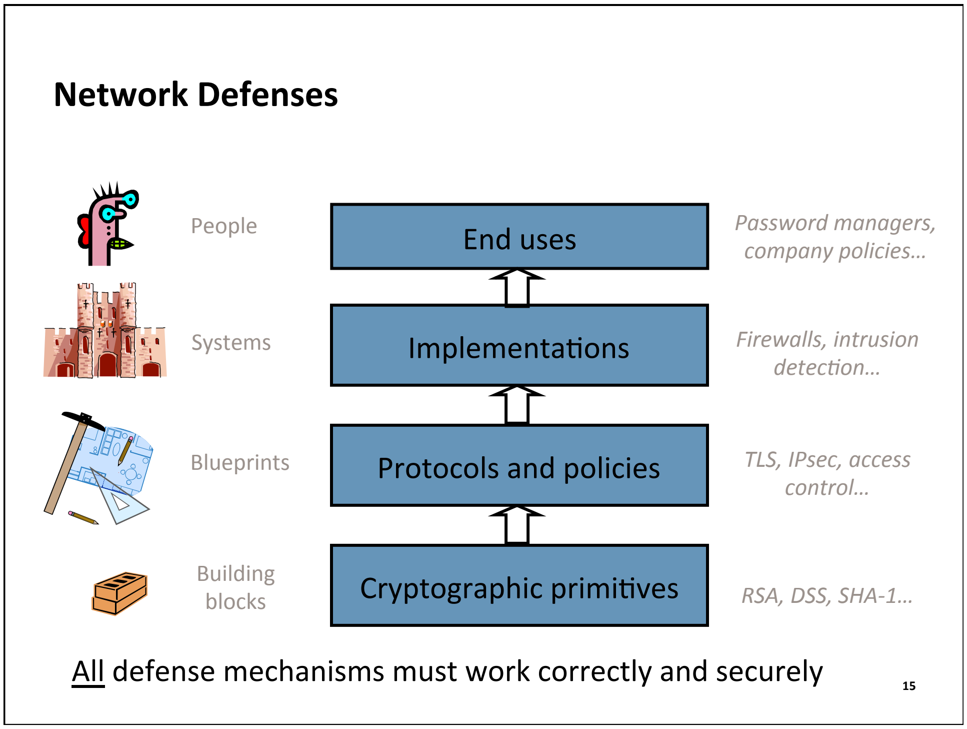
Network Stack



Network stack

people		Phishing attacks, usability
application	email, Web, NFS	Sendmail, FTP, NFS bugs, chosen-protocol and version-rollback attacks
session	RPC	RPC worms, portmapper exploits
transport	TCP	SYN flooding, RIP attacks, sequence number prediction
network	IPv4 / IPv6	IP smurfing and other address spoofing attacks
data link	802.11	WEP attacks
physical	RF	RF fingerprinting, DoS

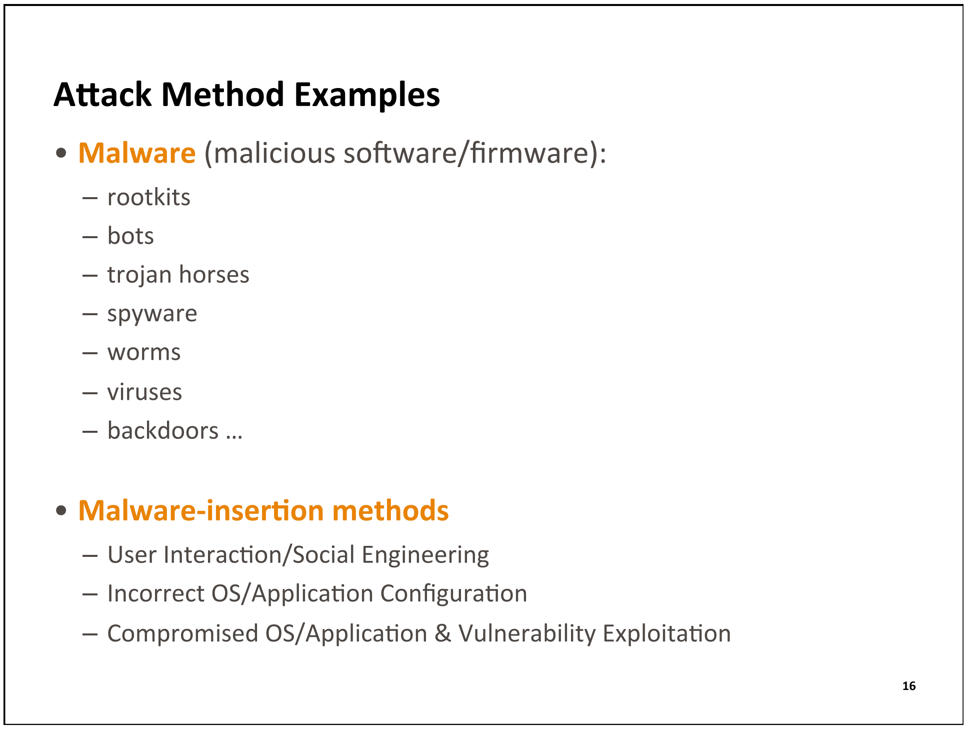
Only as secure as the single weakest layer (or interconnection between layers)



Network Defenses

All defense mechanisms must work correctly and securely

15



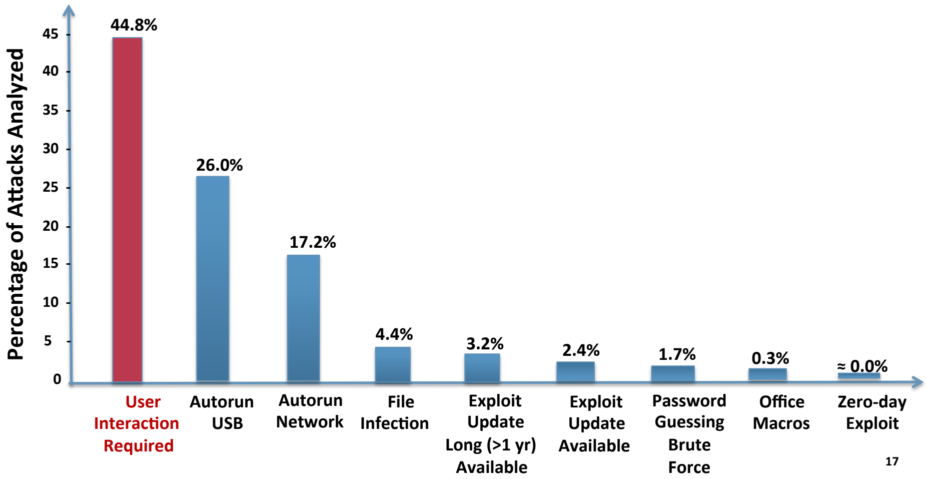
Attack Method Examples

- **Malware** (malicious software/firmware):
  - rootkits
  - bots
  - trojan horses
  - spyware
  - worms
  - viruses
  - backdoors …
- **Malware-insertion methods**
  - User Interaction/Social Engineering
  - Incorrect OS/Application Configuration
  - Compromised OS/Application & Vulnerability Exploitation

16

Malware Insertion Methods

Analysis reported in the **Microsoft Intelligence Report**, **vol. 11, 2011**

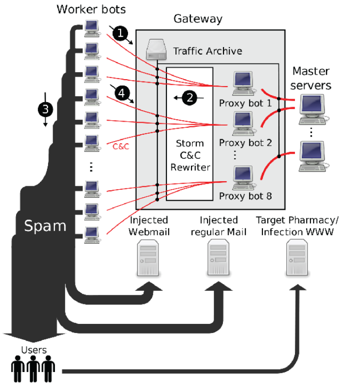


Cybercrime in the Real World

- Botnets
  - **Worker bots** running in the background on millions of compromised hosts
  - **Bot master** sending instructions to worker bots via **command & control** nodes
  - Possible instructions: **propagate**, send **spam**, conduct **DDoS**, **mine Bitcoin**
- Pay-per-Install (**PPI**)
  - "Affiliate" programs rewarding miscreants for installing malware on end-hosts
  - Useful for bootstrapping botnets, sending spam, staging denial of service attacks, performing click fraud, hosting scam websites
- Distributed Denial of Service (DDoS)
  - Instruct a botnet to **direct a large amount of traffic** to the target
  - Leverage protocols that can **amplify traffic** (e.g. NTP, DNS)

Example: Stormbot Spam Architecture

[Kanich, Kreibich, Levchenko et al.]

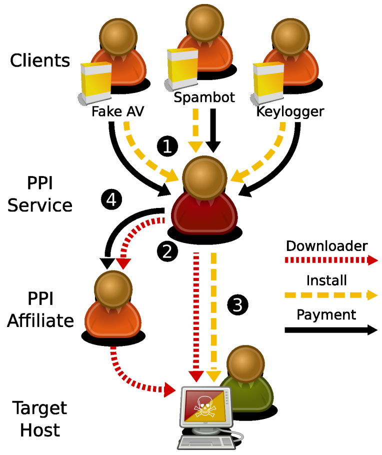


Infrastructure for measuring the activity of the Storm botnet

- Spam templates
  - Custom macro language
  - Polymorphic content
- Dictionaries
  - Email addresses
  - Subject lines
- Worker bots generate unique messages for each address, try to deliver, report results to proxies

Example: The Pay-Per-Install Business Model

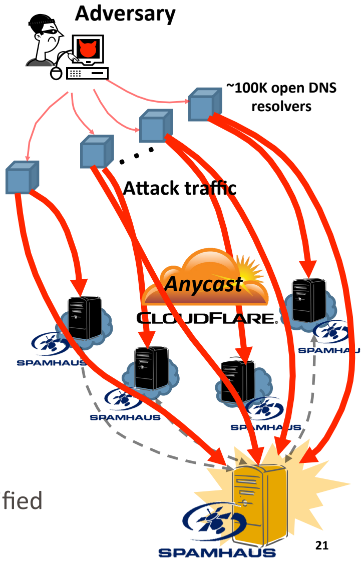
[Cabalero, Grier, Kreibich, Paxson]



1. PPI clients provide software they want installed
2. The PPI service finds affiliates able to provide this service
3. The PPI service pushes the client's executable
4. The affiliates receive commission for successful installations

Example: DDoS Attack on Spamhaus

- Spamhaus provides data on spam-related activities
- In March 2013, it was targeted by a massive DDoS attack
  - 85–120 Gbps on average, over 4 days
  - 300 Gbps peak
- Attack mechanism
  - Attacker sends query for large DNS record to several open DNS resolvers
  - Spoofs IP address, so that replies are sent to the target
  - request << reply => traffic is amplified



Desirable Security Properties

- Authenticity
- Confidentiality
- Integrity
- Availability
- Accountability and non-repudiation
- Access control
- Privacy

…

Correctness versus Security

- System **correctness**: system satisfies specification
  - For reasonable input, get reasonable output
- System **security**: system properties preserved in face of attack
  - For unreasonable input, output not completely disastrous
- Main difference: **intelligent adversary trying to subvert system and to evade defensive techniques**

ENEE 657 In A Nutshell

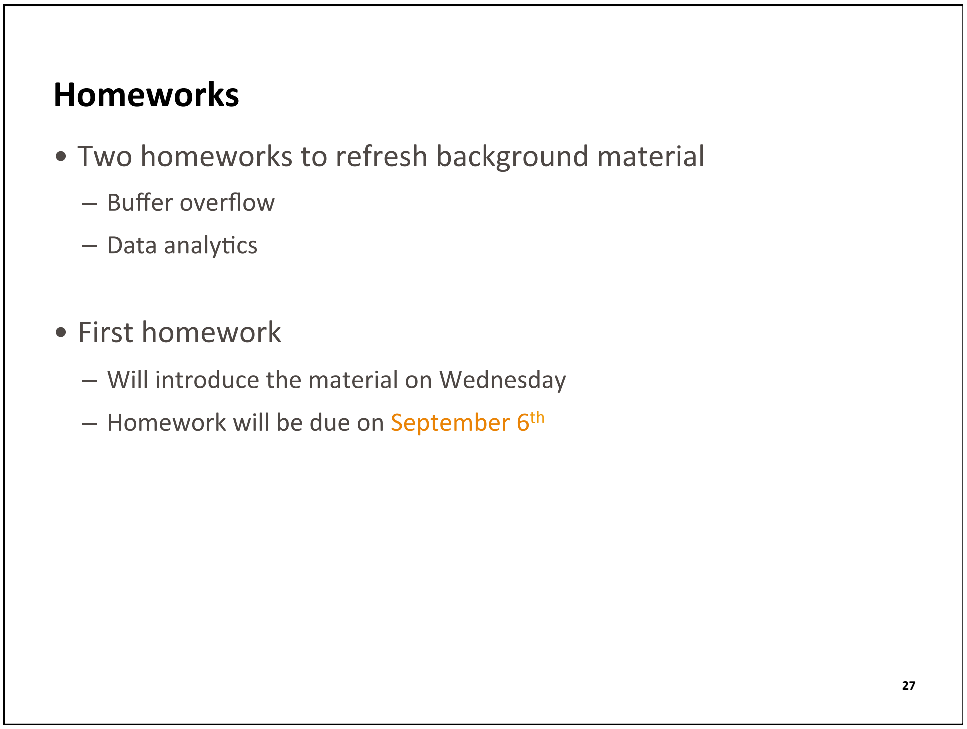
- Course objectives
  - Understand **attacks** and **defenses** in distributed systems
    - To create effective security mechanisms, you must understand the capabilities of **real-world attackers**
  - Prepare you to collaborate with **security researchers**
    - Learn how to **discuss** security topics intelligently
    - Gain thorough grounding in the **techniques** for defending against attacks on distributed systems and networks
- What ENEE 657 is not
  - A course on cryptography
  - A course on theoretical security

ENEE 657 Course Content

- Topics
  - Design and implementation of **protection mechanisms**
    - Vulnerability exploits and defenses against exploitation
    - Privilege separation
    - Confinement
    - Trust and reputation
    - …
  - Security analytics (e.g. **measure** effectiveness of defenses, **infer** malicious activity)
    - Cybercrime measurements (spam, zero-day attacks)
    - Cyber conflict
    - Predicting security events
    - …
- This is a systems-oriented course
  - **Semester-long project**: substantial programming component
  - Project goal: **depth** and **quality** adequate **for publication in a workshop** associated with USENIX Security

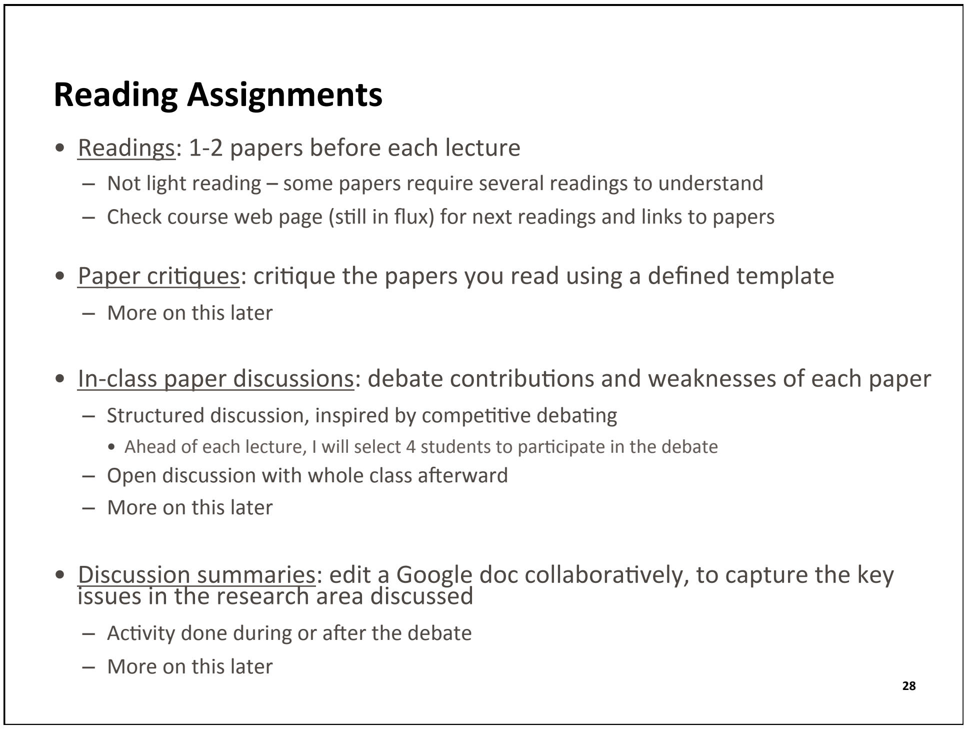
This is a Graduate Course

- Learning the material in this course requires participation
  - This is not a sit-back-and-listen kind of course
  - Understanding the assigned readings is required for understanding the topics
  - In-class discussions are part of your grade
- You are responsible for holding up your end of the educational bargain
  - I expect you to attend classes and to complete reading assignments
  - I expect you to try things out for yourself
  - I expect you to know how to find research literature on security topics
    - The required readings provide starting points
  - I expect you to manage your time
    - In general there will be assignments due before each lecture



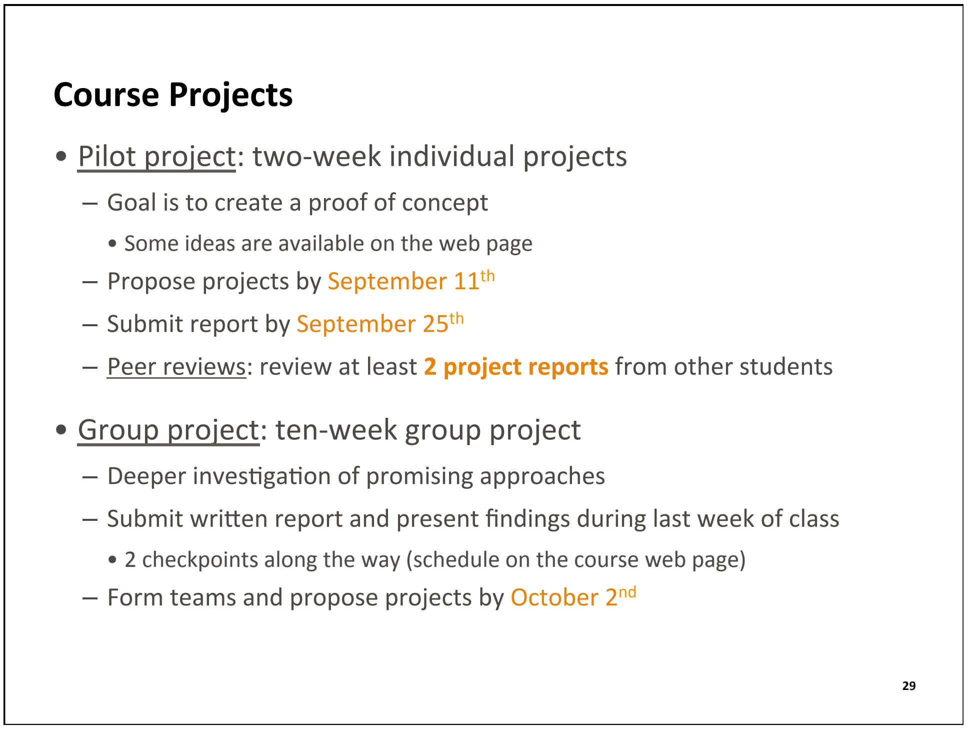
Homeworks

- Two homeworks to refresh background material
  - Buffer overflow
  - Data analytics
- First homework
  - Will introduce the material on Wednesday
  - Homework will be due on September 6th



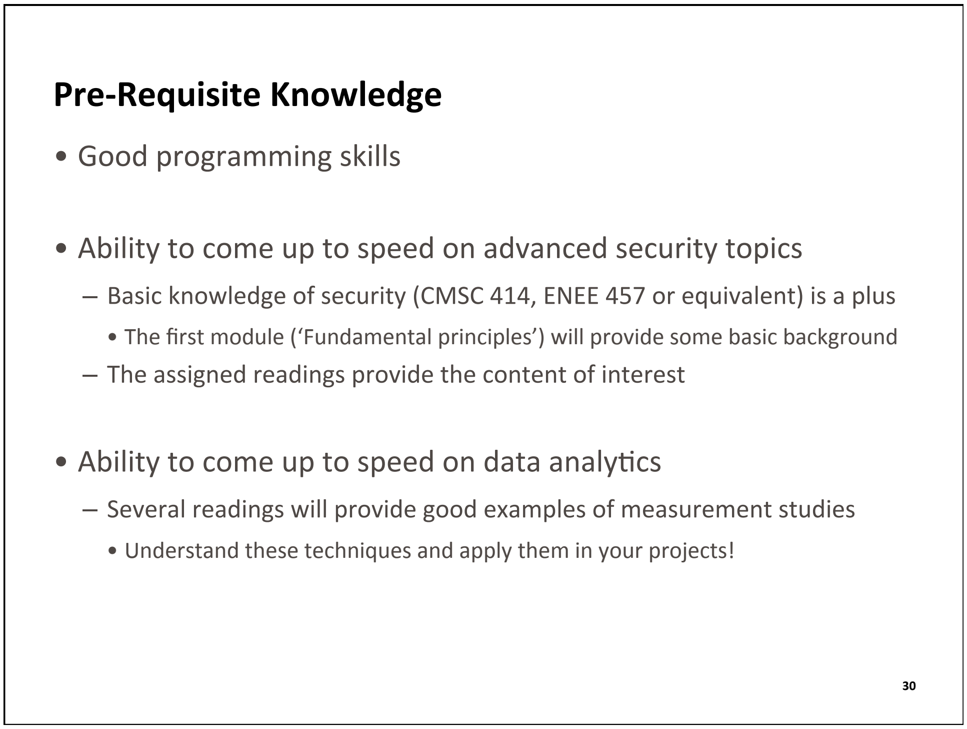
Reading Assignments

- Readings: 1-2 papers before each lecture
  - Not light reading – some papers require several readings to understand
  - Check course web page (still in flux) for next readings and links to papers
- Paper critiques: critique the papers you read using a defined template
  - More on this later
- In-class paper discussions: debate contributions and weaknesses of each paper
  - Structured discussion, inspired by competitive debating
    - Ahead of each lecture, I will select 4 students to participate in the debate
  - Open discussion with whole class afterward
  - More on this later
- Discussion summaries: edit a Google doc collaboratively, to capture the key issues in the research area discussed
  - Activity done during or after the debate
  - More on this later



Course Projects

- Pilot project: two-week individual projects
  - Goal is to create a proof of concept
    - Some ideas are available on the web page
  - Propose projects by September 11th
  - Submit report by September 25th
  - Peer reviews: review at least **2 project reports** from other students
- Group project: ten-week group project
  - Deeper investigation of promising approaches
  - Submit written report and present findings during last week of class
    - 2 checkpoints along the way (schedule on the course web page)
  - Form teams and propose projects by October 2nd



Pre-Requisite Knowledge

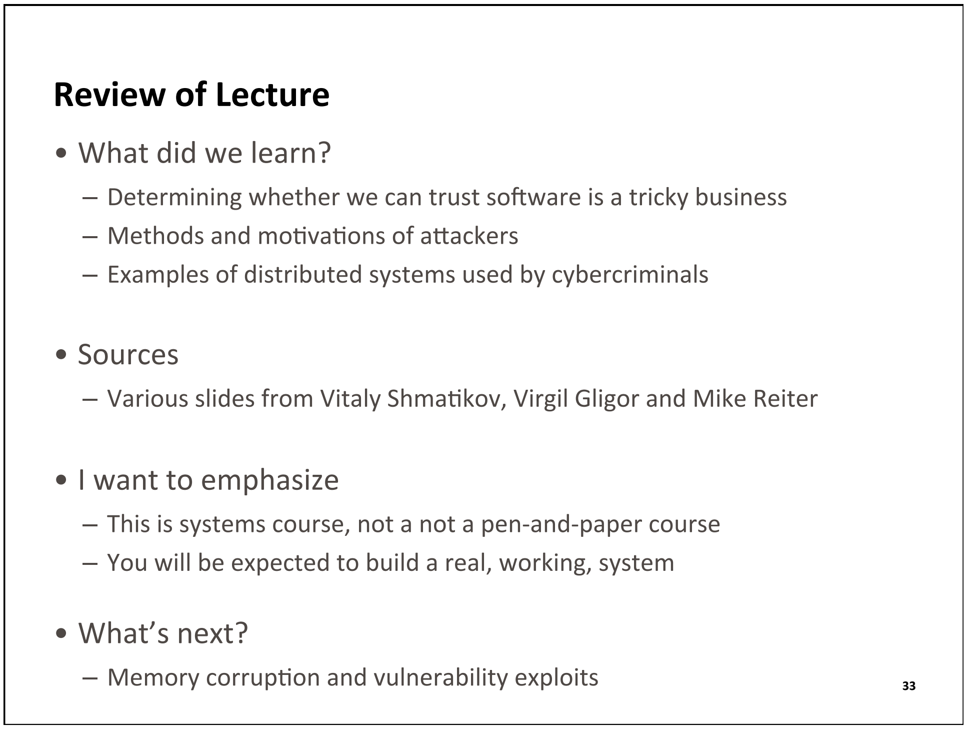
- Good programming skills
- Ability to come up to speed on advanced security topics
  - Basic knowledge of security (CMSC 414, ENEE 457 or equivalent) is a plus
    - The first module ('Fundamental principles') will provide some basic background
  - The assigned readings provide the content of interest
- Ability to come up to speed on data analytics
  - Several readings will provide good examples of measurement studies
    - Understand these techniques and apply them in your projects!

Policies

- "Showing up is 80% of life" – Woody Allen
  - You can get an "A" with a few missed assignments, but reserve these for emergencies (conference trips, waking up sick, etc.)
  - Notify the instructor if you need to miss a class, and submit your assignment on time
- UMD's Code of Academic Integrity applies, modified as follows:
  - Complete your critiques entirely **on your own**. **After** you hand in your critiques, you are welcome (and encouraged) to discuss them with others
  - **Discuss** the problems and concepts involved in the project and homeworks, but **produce your own** implementations
    - Group projects are the result of team work
    - You can post code snippets on Piazza (e.g. to ask a question), but don't post the whole program listing
- See class web site for the official version

Grading Criteria

- Components of the grade
  - 5% Background homework
  - 25% Written paper critiques
  - 30% Participation (in-class discussion, contributions to topic summaries)
  - 40% Projects
  - 10% Potential bonus points
- Expectations
  - You must do **all** the required readings
  - You can explain the **contributions** and **weaknesses** of the papers you read
  - You produce a **working implementation** for your project, and you must **understand** how the implementation works



Review of Lecture

- What did we learn?
  - Determining whether we can trust software is a tricky business
  - Methods and motivations of attackers
  - Examples of distributed systems used by cybercriminals
- Sources
  - Various slides from Vitaly Shmatikov, Virgil Gligor and Mike Reiter
- I want to emphasize
  - This is systems course, not a not a pen-and-paper course
  - You will be expected to build a real, working, system
- What’s next?
  - Memory corruption and vulnerability exploits

33