Network & Distributed System Security (NDSS)
ENEE 757 | CMSC 818V

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

http://ter.ps/757
https://www.facebook.com/SDSAtUMD

About Me

Tudor Dumitraș
Office: AVW 3425
Email: tdumitra@umiacs.umd.edu
Course Website: http://ter.ps/757
Office Hours: Wed 12:30–1:30 pm
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 Big Data experiments in security
  – WINE currently used by academic researchers and Symantec engineers

• Joined UMD faculty

• Research and teaching on applied security and systems
  – Focus on solving security problems with data analysis techniques

NDSS in a Nutshell

• ENEE 757 / CMSC 818V 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 2016

• 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
Who Can You Trust?

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?

“Reflections on Trusting Trust” – cont’d

- Who wrote the compiler?

- Compiler looks for source code that looks 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?
“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
... c = next( );
if(c != '\\')
    return(c);
c = next( );
if(c == '\\')
    return('\');
if(c == 'n')
    return('
');
if(c == 'v')
    return(11);
...```

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

```
... In future versions of the compiler: use the special character
```

“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

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

People

End uses
Password managers, company policies...

Systems

Implementations
Firewalls, intrusion detection...

Blueprints

Protocols and policies
TLS, IPsec, access control...

Building blocks

Cryptographic primitives
RSA, DSS, SHA-1...

All defense mechanisms must work correctly and securely

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
Malware Insertion Methods

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

Distribution of Malware Insertion Methods

- **User Interaction Required**
  - 44.8%
- **Autorun USB**
  - 26.0%
- **Autorun Network**
  - 17.2%
- **File Infection**
  - 4.4%
- **Exploit Update Long (>1 yr) Available**
  - 3.2%
- **Exploit Update Available**
  - 2.4%
- **Password Guessing Brute Force**
  - 1.7%
- **Office Macros**
  - 0.3%
- **Zero-day Exploit**
  - ≈ 0.0%

Distributed Infrastructures Supporting Cybercrime

- **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.]

- 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**
[Cabalaro, 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

NDSS 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 NDSS is not
  - A course on cryptography
  - A course on theoretical security
**NDSS Course Content**

- **Topics**
  - Design and implementation of **protection mechanisms**
    - Authentication and access control
    - Vulnerability exploits and defenses against exploitation
    - Public key infrastructures
    - ...
  - Security analytics (e.g. **measure** effectiveness of defenses, **infer** malicious activity)
    - PKI and cybercrime measurements
    - Botnet infiltration
    - Reputation based security
    - ...

- **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 9th

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
  - First critiques due on September 9th

- 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
**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 14th**
  - Submit report by **September 28th**
  - 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 5th**

**Pre-Requisite Knowledge**

- Good programming skills

- Ability to come up to speed on advanced security topics
  - Basic knowledge of security (CMSC 414, ENEE 459C or equivalent) is a plus
  - Lectures 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
  – 20% Written paper critiques
  – 20% In-class discussion and participation
  – 30% Projects
  – 25% Final exam

• 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

Dive In

http://ter.ps/757