- **If you choose to do a survey project or a project in the “other projects” category, please email me 3 project topics you are interested in by March 6th, indicating your first, second, and third choices.**
- **If you choose to do a programming project, please do NOT email me**

Survey Project
Presentation: April/May
Presentation slides + survey report submission: 11:59pm May 11th

The survey project consists of two parts: (1) present 1 paper, and (2) search the web for 2 papers that relate to (but different from) the paper presented and write a survey report. The survey project is done individually. Each student should give a powerpoint presentation in April or May (around 30 minutes). You will receive an email from me at least 2 weeks before your presentation. The survey report is due on May 11th.

1. Guest-Transparent Prevention of Kernel Rootkits with VMM-based Memory Shadowing
http://www.csc.ncsu.edu/faculty/jiang/pubs/RAID08_NICKLE.pdf

2. Managing Security of Virtual machine Images in a Cloud Environment.
http://portal.acm.org/ft_gateway.cfm?id=1655021&type=pdf&coll=ACM&dl=ACM&CFID=15151515&CFTOKEN=6184618

3. ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay

4.WebCapsule: Towards a Lightweight Forensic Engine for Web Browsers

5. An Empirical Study of dangerous Behaviors in Firefox extensions
www.comp.nus.edu.sg/~xdong/papers/isc12.pdf

6. Tracking the trackers: fast and scalable dynamic analysis of web content for privacy violations
http://www.comp.nus.edu.sg/~liangzk/papers/acns12.pdf

7. An Empirical Study of Privacy-Violating Information Flows in JavaScript Web Applications.

8. Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis.

9. Analyzing Information Flow in JavaScript-based Browser Extensions.

10. An Analysis of Private Browsing Modes in Modern Browsers
http://www.collinjackson.com/research/private-browsing.pdf

11. I Still Know What You Visited Last Summer: Leaking Browsing History Via User Interaction and Side Channel Attacks.

12. Signing Me onto Your Accounts through Facebook and Google: a Traffic-Guided Security Study of Commercially Deployed Single-Sign-On Web Services
http://research.microsoft.com/pubs/160659/websso-final.pdf

13. Data lifetime is a systems problem

14. Shredding your garbage: reducing data lifetime through secure deallocation

15. Eliminating the Hypervisor Attack Surface for a More Secure Cloud
http://www.cs.princeton.edu/~jrex/papers/ccs11.pdf

16. I Know Where You've Been: Geo-Inference Attacks via the Browser Cache.

17. A Security Analysis of Amazon's Elastic Compute Cloud Service
http://www.iseclab.org/people/embyte/papers/securecloud.pdf

18. Browser Privacy Feature: A work in progress + securing your web browser
https://www.cdt.org/privacy/20090804_browser_rpt_update.pdf
http://www.us-cert.gov/publications/securing-your-web-browser

19.Toward Black-Box Detection of Logic Flaws in Web Applications
http://www.internetsociety.org/sites/default/files/02_5_0.pdf

20.Detecting Logic Vulnerabilities in E-commerce Applications
http://www.internetsociety.org/sites/default/files/04_4_1.pdf

21.Parking Sensors: Analyzing and Detecting Parked Domains

22. Vulnerability Statistics for e-banking System
http://www.ptsecurity.com/download/BankingWP.Fv4.pdf

23. Seven Months' Worth of Mistakes: A Longitudinal Study of Typosquatting Abuse

24. I Do Not Know What You Visited Last Summer: Protecting users from stateful third-party web tracking with TrackingFree browser

25. Bloom Cookies: Web Search Personalization without User Tracking

26. Run-time Monitoring and Formal Analysis of Information Flows in Chromium

27. JaTE: Transparent and Efficient JavaScript Confinement

28. Cross-site Framing Attacks

Guideline (ppt slides)

- Motivation: E.g. Why role-based access control?
- Background: E.g. Syntax of xml
- Technical details (use examples/pictures to illustrate technical details)
- Related work

Guideline (presentation)

- Present slowly and clearly, and stop to ask if other students have any questions
- Do not directly read everything from slides
- Use examples to illustrate technical details

Guideline (Survey report)

❖ Provide a 1-2 page summary for each of the papers (does not include the paper presented in the class).
    ➢ What is the problem addressed in the paper?
    ➢ Do you think the problem is important? Why?
    ➢ How does the paper address the problem? – provide a summary of technical details
    ➢ What are the contributions of the paper?
    ➢ What are the strength and weakness of the paper
❖ Your approach, if there is any
❖ Any other points you want to make
❖ Clearly written using your own words.

Programming Project (C/C++/C#/Java)
No presentation
Code submission deadline: 11:59pm May 11th (Wed)

The project will be done by a group of 2 students. At the end of the semester, you will need to submit your code through blackboard. You can use the existing implementations of RSA and SHA1 (e.g. functions provided in java.security and openssl, etc), if necessary. You are not required to implement a graphical user interface (there will be no extra credits if you implement a graphical user interface). If you choose to do the programming project by yourself, you will get 10 points extra credits.

Secure Purchase Order (CS558)

Implement a secure purchase order system that allows a customer to purchase goods online securely. The customer purchases goods from a purchasing server. Once the customer selects goods and enters the credit card number, the purchasing server contacts the bank to verify the credit card number. After the bank verifies the credit card number, the bank sends the result to the purchasing system and updates the credit card balance. The purchasing system will then process the order.

Both the purchasing server and the bank server are iterative servers. The online purchasing server maintains a file **"item"** that stores items sold and the corresponding information about the item. The format of the file is <item #, item name, price, quantities>. Initially the file contains the following contents:

1, lego, $20, 2
2, tv, $800, 1
3, laptop, $1100, 3
4. water, $10, 10

The bank server maintains a file **"balance",** which stores the credit card balance of each user. The format of file "balance" is <name, credit-card-number, balance>. Assume that there are two users "alice" and "tom". Initially, file "balance" contains the following contents:

alice, 11111111, 0
bob, 22222222, 0

Each user has an account on the purchasing server. The purchasing server maintains a password file which contains the following ID (plaintext) and the hash of the following password (hashed using SHA1 or MD5):

ID: alice
Password: 1234

ID: tom
Password: 5678

Let **Pua, Put, Pub,** and **Pup** represent the public key of alice, tom, the bank, and the purchase system, respectively, and **Pra, Prt, Prb,** and **Prp** represent the private key of alice, tom, the bank, and the purchase system, respectively. Assume that both alice and tom have the purchasing server's public key **Pup** and the bank's public key **Pub**, the purchasing server has alice and tom's public key **Pua and Put**, and the bank has the purchasing system's public key **Pup**. **The keys can be generated manually and stored on the disk.**

The purchasing server is invoked as:

psystem <purchasing-system-ip><purchasing-system-port><bank-ip><bank-port> (C/C++)
java Psystem <purchasing-system-ip><purchasing-system-port><bank-ip><bank-port> (java)

The bank server is invoked as:

bank <bank-ip> <bank-port> (C/C++)
java Bank <bank-ip> <bank-port> (java)

The client is invoked as:

customer <purchasing-system-ip><purchasing-system-port>
java Customer <purchasing-system-ip><purchasing-system-port>

The detailed steps are given below:

S1: A customer (alice or bob) invokes the client to connect to the purchasing server. The server prompts the customer to provide his/her ID and password.

S2: After the server receives the ID and the password, the server computes the hash of the password and compares the computed password against the hashed password stored in the password file. If two passwords do not match, the server sends a string "error" to the client. The client then prints "the password is incorrect" and prompts the user to enter the password again.

S3: If the password is correct, the purchasing server sends the contents in file "item" to client. The client then displays the contents using the format <item #, item name, price, quantity>, eg

```
1, lego, $20, 2
2, tv, $800, 1
3, laptop, $1100, 3
4. water, $10, 10
```

The client then displays "Please enter the item #". After the customer provides the item #, the client displays "Please enter the quantity".

S4: The client then prompts the customer to enter his/her credit card number. After the customer provides the credit card number, the client encrypts the item # and the quantity entered by the customer using the public-key of the purchasing server (i.e., E(Pup, <item# || quantity>)). A digital signature DS is also generated for the encrypted message. The client also encrypts the customer's name and credit card number using the public key of the bank (i.e., E(Pub, <name || credit card number>). The client sends E(Pup, <item# || quantity>), DS, and E(Pub, <name || credit card number>) to the purchasing server.

Note that, to simplify the implementation, we do not use dual signature in S4.

S5: The purchasing server decrypts E(Pup, <item# || quantity>) using the private key of the purchasing server and verifies the signature DS. The purchasing server then sends E(Pub, <name || credit card number>) to the bank.

S6: the bank decrypts E(Pub, <name || credit card number>) using the private key of the bank. If the credit card number is correct, then the bank sends "ok" to the purchasing system and update the credit card balance in file "balance". Otherwise, the purchasing server sends "error" to the client.

S7: If the message received is "ok", then the client displays "we will process your order soon", updates the corresponding quantity in file "item", and terminate. Otherwise, then the client displays "wrong credit card number" and terminate.

Submission guideline

- You need to hand in your **source code, public and private keys generated, a readme,** and a **Makefile** electronically (**do not submit executable code**).
- Write a **README** file (text file, do not submit a .doc file) which contains
  - ✦Name and email address of group members.
  - ✦The programming language you use (C/C++/C#/Java)
  - ✦Platform (Bingsuns/Linux/Windows)
  - ✦How to execute your program.
  - ✦Code for performing encryption/decryption
  - ✦(Optional) Anything special about your submission that the TA should take note of.
- Place all your files under one directory with a unique name (such as proj-[userid] for the project, e.g. proj-pyang).
- Tar the contents of this directory using the following command.

  tar –cvf [directory_name].tar [directory_name]

  E.g. tar -cvf proj-pyang.tar proj-pyang/
- Use the Blackboard to upload the tared file you created above.

Grading Guideline

- ●Readme – 5'
- ●Makefile – 5'
- ●Hashed password - 10'
- ●Encryption/decryption – 25'
- ●Other functionality: -- 55'

Other Projects

Presentation + demo: May 11th (in class)

Submission deadline (source code if applicable + slides): 11:59pm May 11th

Each project is done by a group of 2 students. Each group will give a 20-25 min presentation and show demo in the class on May 11th, describing the design and implementation of the project. If you choose to do the project by yourself, you will get 10 points extra credits. You will also need to submit your code and presentation slides by 11:59pm on May 11th.

1. Buffer Overflow Attack (language: C)

Work through the shell example given in
http://insecure.org/stf/smashstack.html

Note: Use a linux machine (instead of bingsuns) to do this project.

2. Kernel Rootkit II (lauguage: C)

Design and implement a linux kernel rootkit that modifies the system call table to hide the file you create (ls).

3. Virus (language: C)

Design and implement a virus that can infect all executable files under a specific directory. Note that the program can still execute after the program is infected with the virus. When the infected program executes, the virus will execute first, and then the program.