SysSec Summary

Frédéric Vogel `vogelfr@ethz.ch` , ETH Zürich, HS17

Side Channels

Some implementations of provably secure cryptosystems can be broken by observing the side information leaked by these implementations.

Timing cryptanalysis

Cryptosystems vulnerable when

1. *execution time* depends on key *for different input data*, and
2. *partial execution time* can be measured

Protection

Equalizing the time of all operations independently of the input values

But:

- Increases average time of operation
- Difficult to build soft- and hardware that holds property

Masking (Kocher):

- Chose *random* X, different for each message M
- $S = [(mX)^d \mod n] \cdot [(X^{-1})^d \mod n]$
- $= [m^d \mod n] \cdot [(X \cdot X^{-1})^d \mod n] \mod n$
- $= m^d \mod n$

$(X^{-1})^d \mod n$ can be computed in advance

Masking (Rivest):

- Chose *random* Y
- $S = ([m(Y^{-1})^e]^d \mod n) \cdot Y \mod n$
- $= [m^d \mod n] \cdot [(Y^{-1})^{ed} \mod n] \cdot Y \mod n$
- $= m^d \mod n$

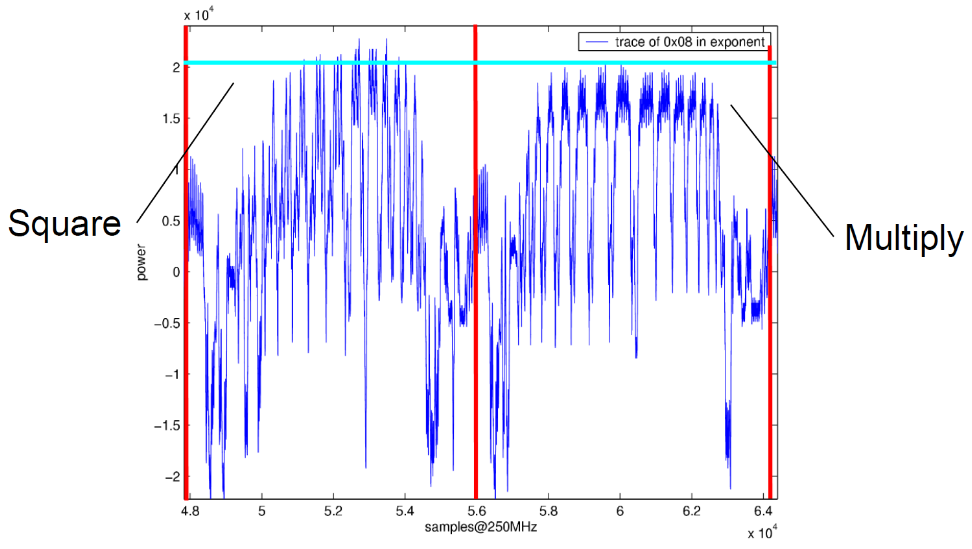
$(Y^{-1})^e \mod n$ can be computed in advance

Power Analysis Attacks

An attack that enables to retreive a secret by observing power consumptions of a device

Simple Power Analysis

Measure power consumption of instruction sequences that depend on the key



If squaring is followed by other squaring that bit is 0, if squaring is followed by multiplication that bit is 1.

Differential Power Analysis

Power consumption depends not only on the type of executed instruction but also on values of operands:

- Storing data to register or memory (storing 0 *vs* storing 1)
- Shifts and rotations
- Logical and arithmetic operations

E.g. attack on DES: Half of key is in C register and rotated each round in deterministic way.

Protection

1. Desynchronization
2. Noise generator
3. Filter at the power input + physical shielding
4. Software balancing
5. Hardware balancing

Cache timing attacks

Table lookups/memory addresses are derived from key+plaintext. Depending on where the data is (cache, memory) access time will vary leaking information.

Block ciphers

- Data in blocks of N bits
- Each block seen as symbol (Alphabet size: 2^N)
- Block of plaintext mapped to block of ciphertext ($2^N!$ mappings)
- Secret key indicates which mapping to use

Problem: Key to big for ideal block cipher ($\log_2(2^N!) bits$, $10^{11} GB$ when $N = 64\}$)

Solution: Key of K bits to specify random subset of 2^K mappings

Shannon's Confusion and Diffusion Principle

Diffusion
Ciphertext bits should depend on plaintext bits in complex way (if M_i is changed C_i should change with $p = 0.5$)

Confusion
Each bit of C should depend on the whole key (change of one bit in the key changes entire ciphertext)

Attack on AES

1. Offline phase:
   For each byte of the key one index value will have slowest lookup time, find value for each byte (e.g. k[13] is slowest when index value is 8)
2. Attack phase:
   - Measure time needed by server to encrypt each sent message
   - Observe that average AES time is maximum when e.g. $n[13] = 147 \Rightarrow k[13] \oplus 147$ takes longest
   - Compare with value from offline phase (k[13] \oplus 147 = 8 \Rightarrow k[13] = 155)

Problem: Difficult to write *constant*, *high-speed* AES Software
Solution: Special AES instruction on chip decouples AES from cache

Transmitted Electro-Magnetic Pulse / Energy Standards & Testing (TEMPEST)

Compromising emanations generated by any electrical information processing equipment can be sensed over air, water, electrical lines and many more

Tamper Resilience

Tamper Resistant
Prevention of break-in

Tamper responding
Detection of intrusion with appropiate response

Tamper evident
If intrusion occurs, evidence of it is left behind

Smart-cards

- Memory read-out by physical access to circuitery during operation
- Glitch attack try to induce a malfunction in the device, e. g. to ignore loop variable decrement in order to dump entire memory
- Most vendors use hardware and software countermeasures to protect against attacks.

Protocol-level/API Attacks

- Unrelated to side-channels
- **Can we trick the HSM into leaking secrets by sending the right requests?**
- Assumption: Can only query API, no tampering

Decimalization Attack

1. Change decimalization table $\rightarrow$ figure out existence of single digits in PIN
2. Change offsets $\rightarrow$ figure out position of digit in PIN

Crypto Token Attack

1. Export target key K_1 encrypted with $K_2 \rightarrow$ Output C
2. Tell token to decrypt C with $K_2 \rightarrow K_1$ decrypted

API Attacks can be very subtle and hard to find

Security on Commodity Systems

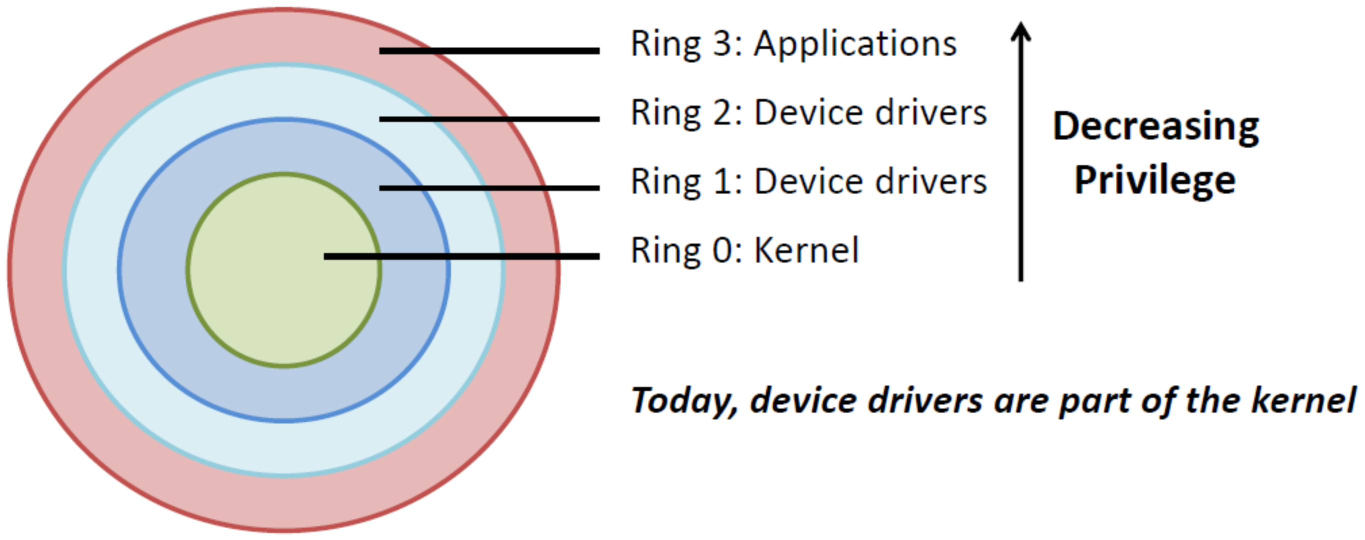
Application Security

Three important properties:

Property	**Functionality**
Launch-time integrity	Integrity (e.g. hash) verification of inital code, data
Run-time isolation	Prevention of unauthorized modification of application code and data Prevention of run-time attacks (e.g. code injection) that modify execution flow
Secure persistent storage	Confidentiality, integrity protection of persistent data

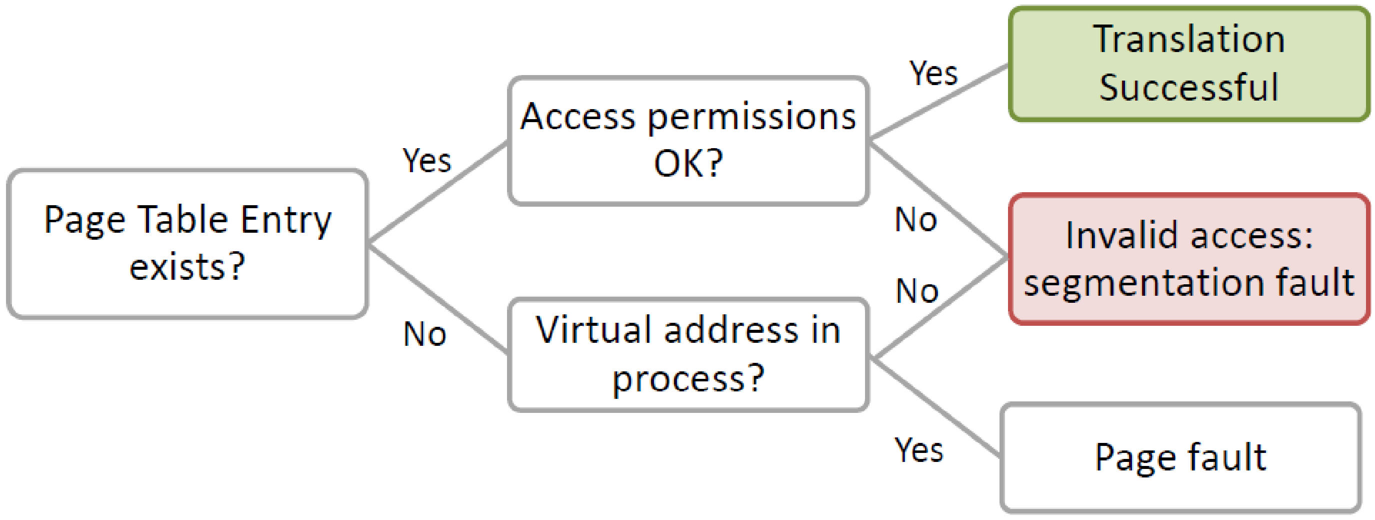
- Where, how to implement this functionality?
- How to protect these security functions themselves?

OS-based Security



Currently only rings 0 and 3 in use

Paging-based Security



Bit	Effect
Supervisor bit	If set, page only accessible from ring 0
Read-Write bit	To distinguish between read-only and writable pages
Execution disabled bit	If set, page is not executable (prevents run-time injection)

Can be circumvented with Direct Memory Access (DMA), e.g. FireWire let’s you access the memory directly and therefore also stuff you are not supposed to see.
Solution: IOMMU (Input-Output Memory Managment Unit) between DMA device and Memory, same function as MMU between CPU and memory.

Attacks using Physical Access

Physical Access Attacks are harder to defend for the OS:

1. Remove hard drive from unattended machine
2. Boot from other machine and copy data

BIOS Protection

Prevent booting from external sources in BIOS, protect BIOS with password
Broken:

- BIOS Reset jumper clears password
- Removing battery for period of time resets BIOS

Disk encryption

Entire key is encrypted and unlocked by providing secret key

Password only	Secure Element
OS asks user a password on boot	OS asks user a password on boot
OS unlocks the encryption key using password	OS uses password to unlock encryption key from TPM
OS places key in memory and uses it to decrypt disk	OS places key in memory and uses it to decrypt disk

Problem: DRAM content not immediately lost when power gone
Attack: Access RAM, freeze RAM, read RAM content from acquisition platform, recover key

Overview

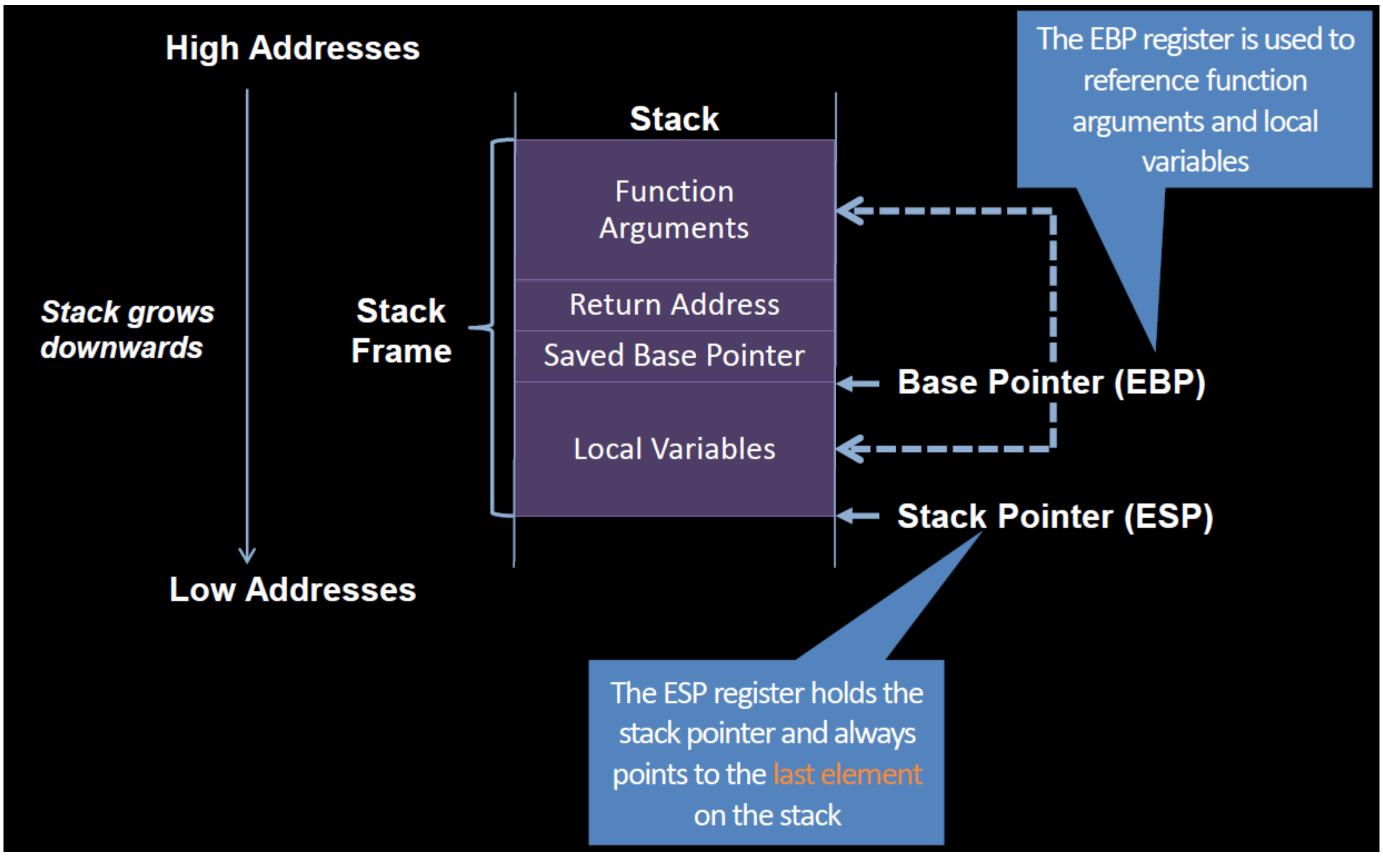
	Hardware support for OS-based Security
CPU	Privilege rings Memory Managment Unit
Chipset	DMA Remapping tables (IOMMU)
Peripherals	Trusted Platform Module Normal HDD with OS-enforced access control

Application Security: Trusted Execution Environments (TEE)

> **ToDo**

Modern Runtime Attacks and Defenses

Each function is associated with one stack frame on the stack


High Addresses
Stack grows downwards
Low Addresses
Stack
Stack Frame
Function Arguments
Return Address
Saved Base Pointer
Local Variables
Base Pointer (EBP)
Stack Pointer (ESP)
The EBP register is used to reference function arguments and local variables
The ESP register holds the stack pointer and always points to the last element on the stack

Code Injection Attack

1. Buffer overflow
2. Code injection
3. Control-flow deviation

Solution: Data Execution Prevention (DEP)

Prevent exectuion from writeable memory (data) area (enabled by default on modern OSes)

Code Reuse Attack

1. Buffer overflow
2. Control-flow deviation (towards existing code)

What code to reuse?

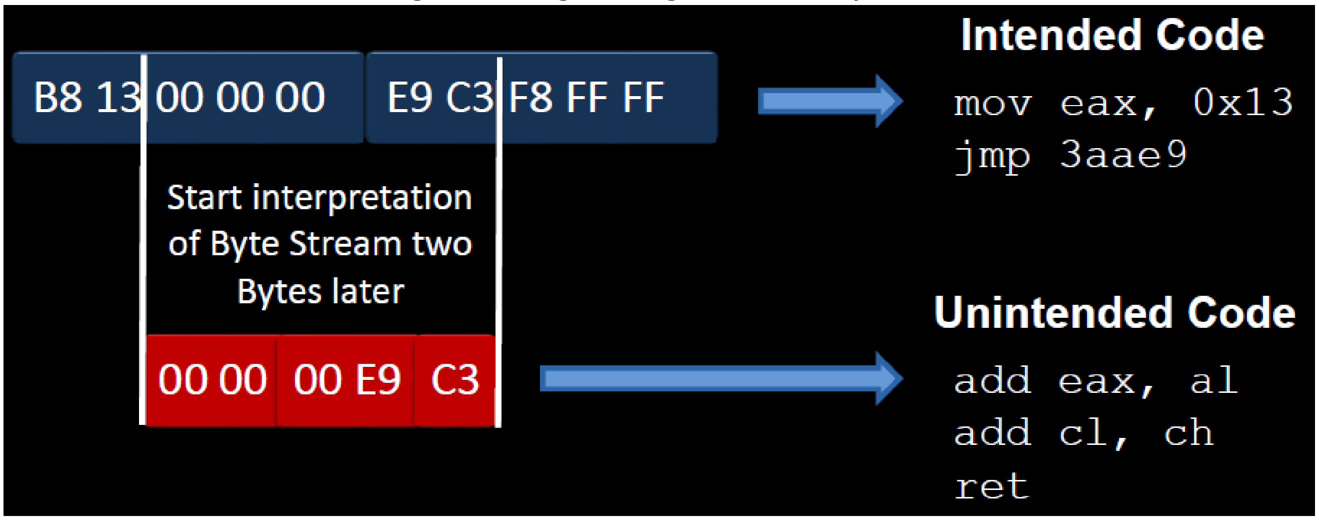
Libraries, mainly `libc`

`libc`
linked to nearly all Unix program
Defines system calls and functionalities such as `open()` , `malloc()` , `printf()` , `system()`
`system("/bin/sh"), exit()`

Solution: Address Space layout Randomization (ASLR)
ASLR randomizes base address of code/data segments

Return-oriented Programming (ROP)

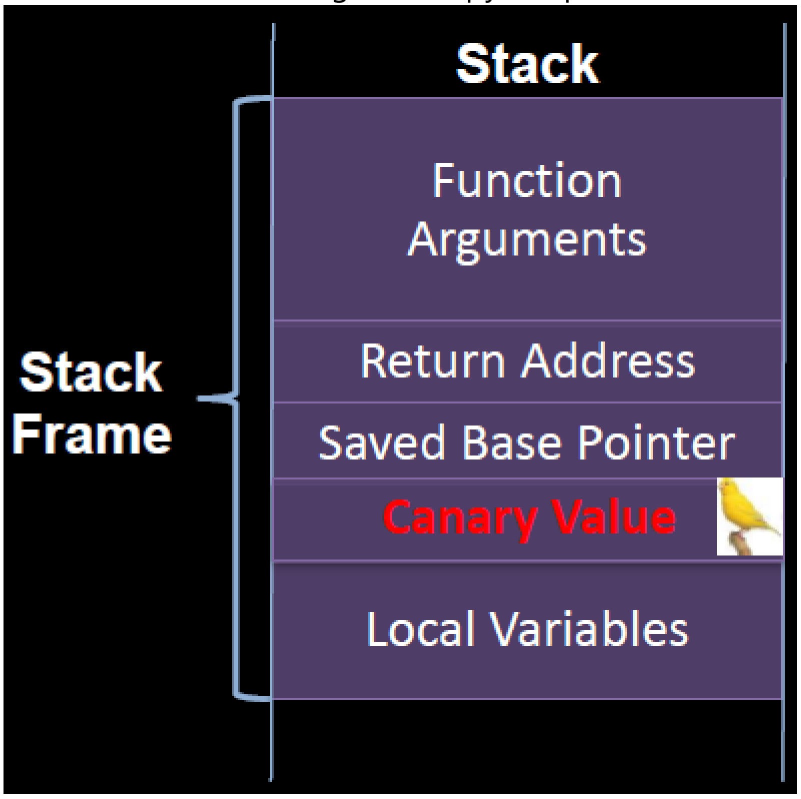
- Perform arbitrary computations with gadgets
  - Use *small instruction sequences* (2-5 instr) *terminated by return instruction* instead of while functions
  - Chain instruction sequences together to form *gadget*
  - Single gadget performs *particular task*
  - Arbitrary computation by *combining* gadgets
- Get new code out of existing code (e.g. unaligned memory access)



- *Turing complete*

Possible defense: Stack Canaries

Write random value onto stack during function prologue and story copy "far away"
Check random value against copy, stop on mismatch



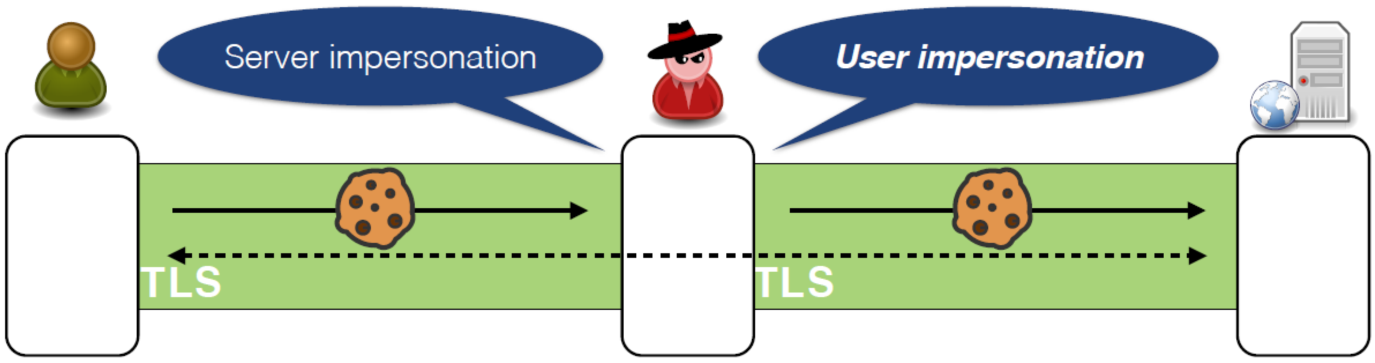
Possible defense: Shadow Stack

CPU protects return addresses in special, separate memory

Online Authentication (Selected Issues)

TLS Man-In-The-Middle (MITM)

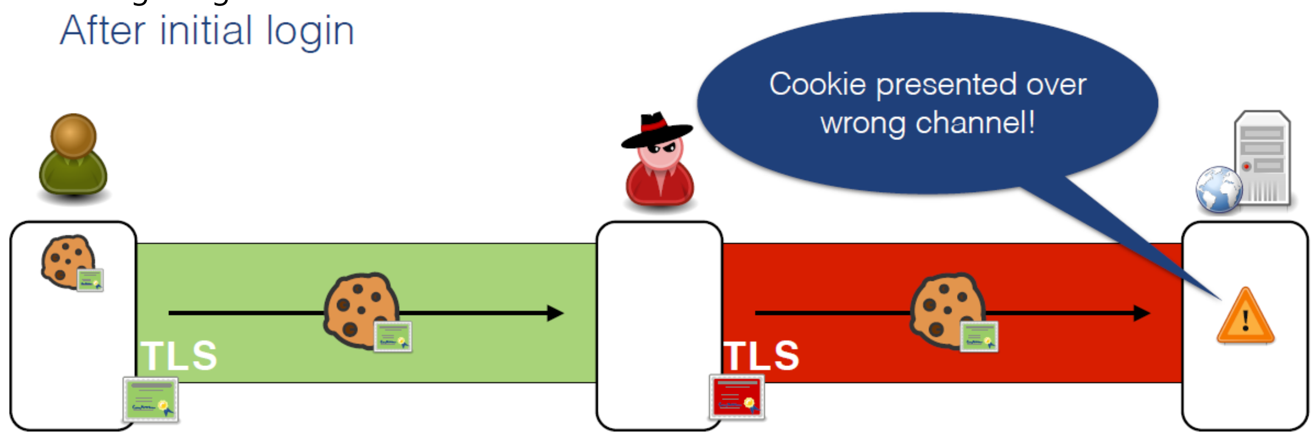
Goal: compromise user account



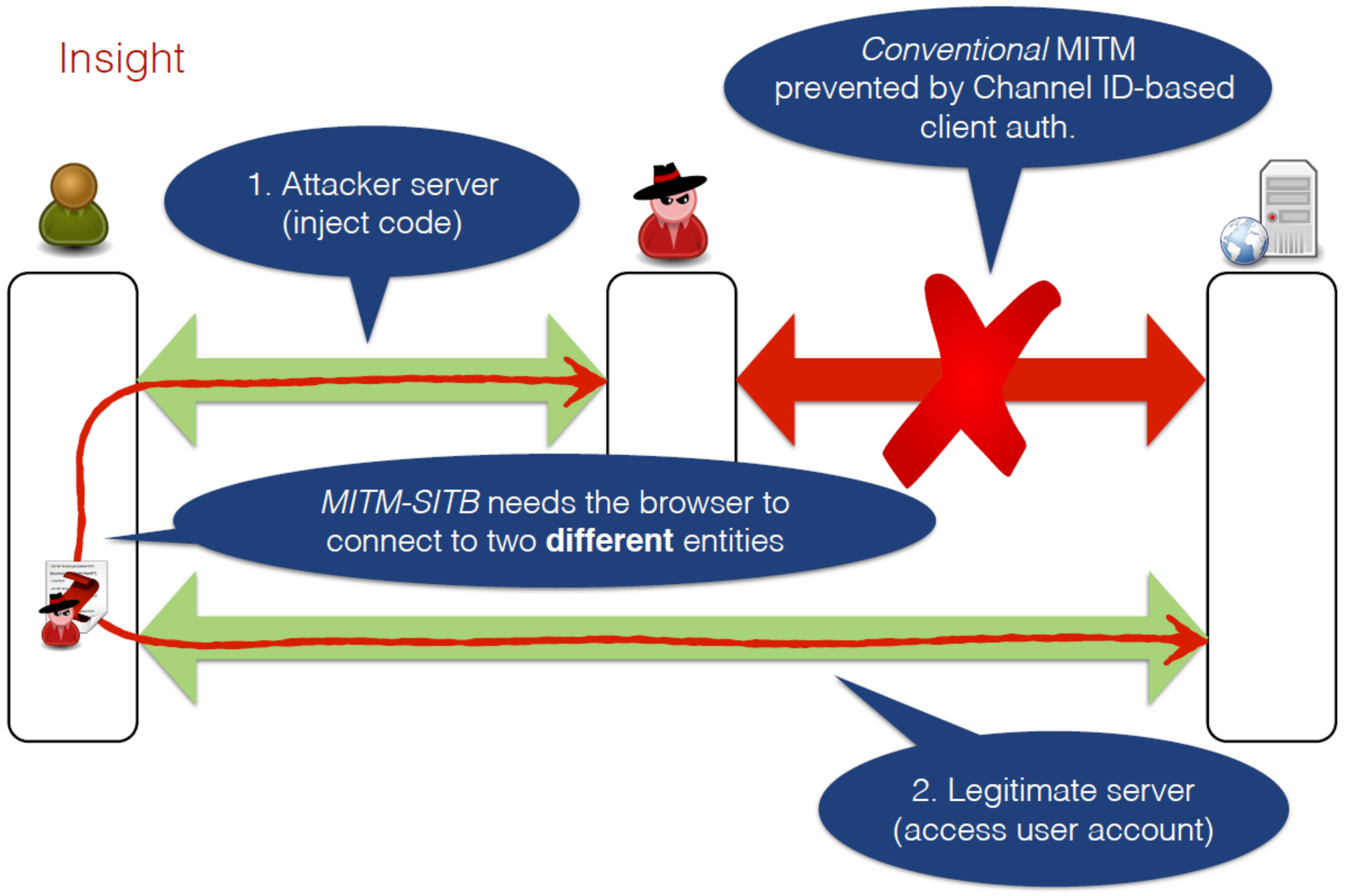
Possible solution: TLS Channel IDs

Channel ID = public key of a private/public key pair
A Channel ID identifies the same “TLS channel” across different TLS connections

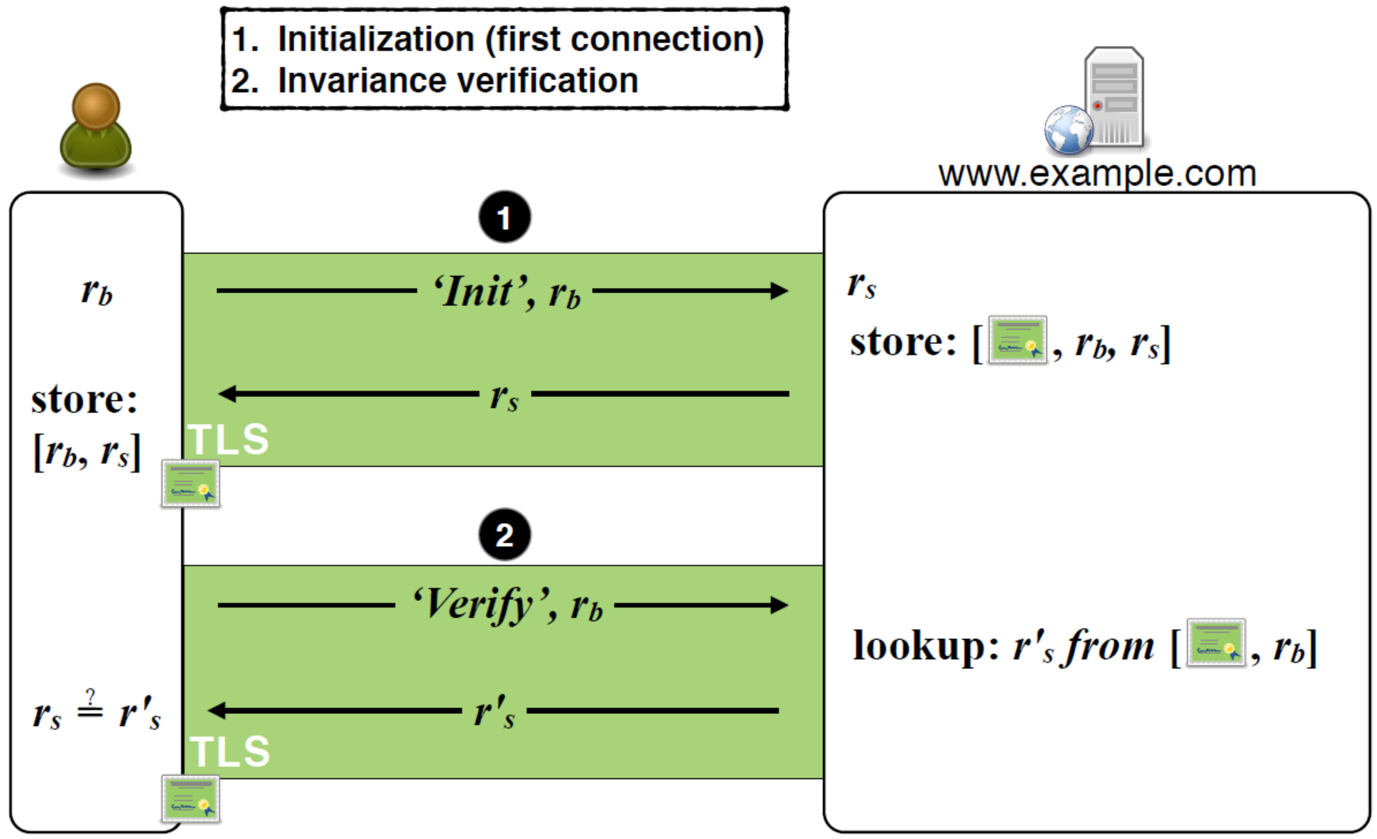
Cookies get signed with Channel ID → Channel-bound cookies



MITM-Script-In-The-Browser (MITM-SITB)


Insight
Conventional MITM prevented by Channel ID-based client auth.
1. Attacker server (inject code)
MITM-SITB needs the browser to connect to two **different** entities
2. Legitimate server (access user account)

Solution: **Server Invariance with Strong Client Authentication (SISCA)**



1. Initialization (first connection)
2. Invariance verification
www.example.com
1
2
r_b
'Init', r_b
r_s
store: $[r_b, r_s]$
TLS
r_s
store: [, r_b, r_s]
'Verify', r_b
r'_s
$r_s \overset{?}{=} r'_s$
lookup: r'_s from [, r_b]
TLS

Operating System Security

Secure OS Definition

A reference monitor: System to monitor and enforce a security policy

Complete mediation: All security-sensitive operations must be directed to the reference monitor

Tamperproof: Enforcement mechanism cannot be modified by untrusted processes

Verifiable: Small enough to audit, prove it satisfies goals

Compartmentalization

Qubes OS

- Securely isolated compartments called qubes
- One compromised qube won't affect the others
  - Compromising an application as easy as before but has limited impact
- Xen hypervisor
- One admin domain, several user domains

Different colors:

- different VMs

- Different Trust Levels
- Different File Systems, etc.
- Colors can not be forged by applications

Disposable Qubes

- Very lightweight VMs, can be created and booted quickly
- E.g. view sensitive but potentially malicious document
    - Sensitive information not leaked
    - Malicious document isolated from anything else

Pro	**Con**
Easily start VMs, Copy-Paste between VMs	May be difficult for (non-expert) users
Protected clipboard, Secure File Transfer	Requires complex configuratio, disciplined use

Gains:

- Reduces attack surface per use case $\rightarrow$ smaller Trusted Computing Base (TCB)
- Limits impact of compromise
- Less inter-compartment interactions $\rightarrow$ easier control

But not designed to be tamperproof and verifiable

Microkernels

Kernel as small as possible:

- Only keep necessary parts in kernel
- Minimal Trusted Computing Base (TCB)
- Reduce attack surface

Essentials:

1. Address space managment
2. Scheduling
3. Inter-Process Communication (IPC)

Per application TCB

- Microkernel
- Needed servers

    - Daemon program
    - Kernel grants privilege to interact with parts of physical hardware
    - Device drivers interact directly with hardware
- **IPC performance** very important

sel4

- Efficent and high assurance (verifiable)
- Comprehensive *formal verification* down to machine code
- Enforces critical security properties as information flow control, integrity and confidentiality
  Verification
- Automated verification of correctness of compiler-generated code
- High-level abstract specification allows verification of high-level properties
- Full functional correctness (abstract specification describes all possible functional behaviours of system)
- Refinement proof shows behaviour of binary implementation is fully captured by abstract specification

Pro, cons

Pro	Con
Closer to the ideal secure OS definition	Limited functionality, performance
Complete mediation, tamperproof, verifiable	Very expensive to build and update

SELinux

Linux Security Module (LSM) Framework

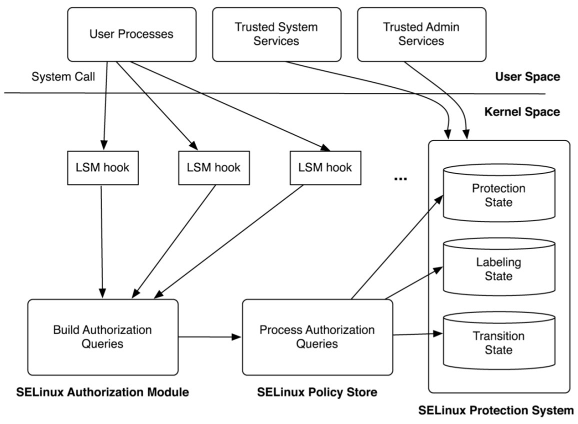
- Hook security-sensitive accesses
- Over 150 hooks defined
- Implementation by various security modules

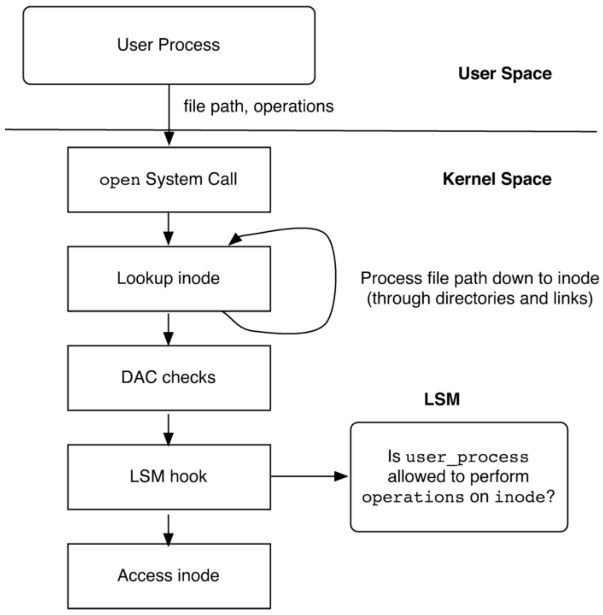
SELinux

Implements LSM hooks

- Mandatory Access Control (MAC) for Linux
    - Opposed to Linux’ standard discretionary access control (DAC)

- Two main components
  - Authorization module: converts input from LSM hooks into authorization queries
  - Policy store: processes authorization queries by looking up policy




User Process
User Space
file path, operations
open System Call
Kernel Space
Lookup inode
Process file path down to inode (through directories and links)
DAC checks
LSM
LSM hook
Is user_process allowed to perform operations on inode?
Access inode

- Files labeled on disk
- Concepts
  - Default deny: all that is not explicitely allowed is denied
  - Confinement: adding user to `user_u` means user cannot use `setuid` applications or execute files inside home directory
  - Separation: cannot compromise one process and use it as vector for another
- Labels: `user:role:type:(level)`
- Fine-grained access control
- Not tamperproof or verifiable

Android

- Isolate applications instead of isolating users
- Each application gets separate Linux UID
  - Apps cannot read each other’s files/memory
  - Apps cannot exhaust all ressources by themselves
- Permissions requested by application in manifest
  - Permissions of type “Dangerous” need user approval
- Applications have to be signed by developer
  - Updates have to be signed with same key
  - Applications signed by same key can request same UID

Overview

- Trend changing from isolating users towards isolating apps
- Least privilege, compartimentalization, isolation, protection of sensitive information
- OS Security is hard

Designing Secure Systems based onTrustworthy Computing and Attestation

Three-step approach:

1. Establish isolated execution environment
    - Ensures partition from untrusted components
    - Hardware ensures partition
2. Externally validate correctness of execution environment
    - Use external root of trust to establish local root of trust
3. Autonomus launch and operation of execution environment
    - After root of trust is set up, use without external validation
    - Sealed storage enables secure local execution after local root of trust is set up

Trusted Platform Module (TPM)

Goals:

- Platform identity
- Remote attestation
- Sealed storage
- Secure counter

Misconceptions:

- TPM is passive, *not* general-purpose processor
- TPM is not tamperproof
- TPM is not part of main processor

Basic functions

- Platform Configuration Registers (PCR): store integrity measurement chain
    - PCR_{new} = SHA-1(PCR_{old} || SHA-1(data))
- On-chip storage for Storage Root Key K^{-1}_{SRK}
- Manufacture certificate
- Remote attestation (PCRs + AIK)
    - Attestation Identity Keys (AIKs) for signing PCRs
- Sealed storage (PCR + SRK)
- Random number generation

Attested Boot (TCG 1.1)

- Measurement of all executed software and configuration files define platform configuration

TO BE FINISHED

Intel Software Guard eXtensions (SGX) / ARM Trustzone

SGX

- Security critical code isolated in enclave
- Only CPU is trusted
- Enclaves can not harm system
- Designed for multi-core systems

Readings

Remote Timing Attacks are Practical (https://crypto.stanford.edu/~dabo/papers/ssl-timing.pdf)

RSA decryption implementations often have branches depending on the input. Those can be used in a side-channel timing attack, even across a network or VMs.

Attack

Timing differences in OpenSSL to calculate $g^d \mod q$ come from two sources:

- Extra reductions in a Montgomery reduction
  As g approaches a multiple of the factor q the number of extra reductions increases.
  When we are just over a multiple of q the number decreases.
  Decryption of $g < q$ is slower than $g > q$.
- Karatsuba *vs* normal multiplication
  When g is just below a multiple of q OpenSSL uses Karatsuba. When g is just over a multiple of q then OpenSSL uses normal multiplication.
  Decryption of $g < q$ is faster than $g > q$.

The key can be found by using the timing results with varying parameters:

- Number of samples for particular ciphertext
  From 5 samples on decryption time converges and has low variance.
- Neighbourhood size n
  For every bit of q the decryption time for a neighbourhood of values $g, g+1, g+2, \ldots, g+n$ is measured

Defences

- RSA blinding:
  $x = r^e g \mod N$ is calculated before decryption where r is random. After x is decrypted normally $x^e/r \mod N$ is calculated to get the cleartext. Since r is random the decryption will take random time.
- Make the decryption time independent of input: use single multiplication mehtod and always perform extra reduction in Montgomery's algorithm.

Cache-timing attacks on AES (http://cr.yp.to/antiforgery/cachetiming-20050414.pdf)

The complete AES key can be recovered from a network server on another computer. The attack works by exploiting a side channel of many AES implementations, namely **S-box lookups**. The index of the lookup is input dependent and since table lookups do not take constant time they leak information.

Skipping an operation is faster than doing it

Input-dependent branches result in different execution time depending on the branch taken.

Cache is faster than DRAM

If parts of the S-box are stored in the cache but not all of it information will be leaked by the lookup time.

- *L1-table-lookup instruction. Same as below*

L1 cache is faster than L2 cache

Similar to previous.

- *L1-table-lookup instruction: (1) ensures entire table is in L1, (2) loads a selected table entry in a constant number of CPU cycles.*

Cache associativity is limited

Depending on the layout of the Cache the different AES operations will kick used AES array out of cache.

- *Compress the S-boxes to fit entirely in cache*
- *If AES input and key must be loaded from uncontrolled location assume AES S-box lines have been kicked out of cache -> reload entire S-box.*

Code can be interrupted

Multithreading prevents any guarantee that no cache line will be kicked out *during* the execution of AES.

- *Include cryptographic software into OS-kernel.*
- *If AES computation is interrupted restart from scratch.*

Stores can interfere with loads

Even when the data is kept in cache loads from a cache line takes slightly more time if it involves the same cache line mod x as a recent store to the cache.

- *Use compressed S-boxes, shift the stack to a position just below the beginning of the AES S-boxes.*

Cache-bank throughput is limited

Loads from cache can bump into each other, taking slightly different amounts of time depending on their addresses.

- *Spread the loads to take place in separate cycles*