A6: Sensitive Data Exposure
A6 – Sensitive Data Exposure

- Sensitive data stored or transmitted insecurely
  - Failure to protect all sensitive data
    - Usernames, passwords, password hashes, credit-card information, identity info
    - Session IDs, cookies, URLs
  - Failure to protect all places sensitive data gets stored
    - Databases, files, directories, log files, backups, etc.
  - Failure to protect all transmissions of sensitive data
    - Web, backend databases, business partners, internal communications
Example: Artifacts in source code

- Developers leaving secrets or tests in code
- Comments by developers giving hints to hidden functionality (within HTML or code).
Example: Insecure Storage

1. Victim enters credit card number in form

2. Error handler logs CC details because merchant gateway is unavailable

3. Logs are accessible to all members of IT staff for debugging purposes

4. Malicious insider steals credit card numbers
Example: Insecure Transport

1. External attacker steals credentials and data off network

2. Internal attacker steals credentials and data from internal network

Target 2013 breach $252 million
Example: Poor use of cryptography

- Weak algorithms (base64, MD5, AES-ECB Mode, RC4/SSL 3.0)
- Poorly used algorithms
  - Pseudo-random number generators (PRNGs) with predictable seeds
  - Unsalted cryptographic hashes (rainbow tables)
- Examples
  - Guessable two-factor PIN codes
  - Guessable password resets (e.g. generated passwords, reset links)
A6 – Prevention
Verify architecture

- Ensure threat model accounts for possible attacks
  - All sensitive data
  - All the places that data is stored
  - All times that data is communicated
Use algorithms appropriately

- Use standard strong algorithms
- Verify
  - All keys, certificates, and passwords are securely generated, distributed, stored, and protected
  - Effective plan for key change are in place
  - Audit code the utilizes encryption code for common flaws
    - (e.g. unsalted password hashes, uninitialized data)
Enable transport security

- Enable TLS for all connections
  - HSTS (HTTP Strict Transport Security)
  - HSTS Chrome preload list
- Employ certificate and public key pinning
  - Key continuity to prevent rogue CA from redirecting your traffic
  - WoSign 8/2016
- Disable vulnerable algorithms
  - Vulnerable SSL versions (Poodle)

http://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet
Labs and homework

- Toy examples that don’t require topics in CS 485/585 to perform
  - For more, take CS 485/585
  - Do the Matasano crypto challenges [http://cryptopals.com](http://cryptopals.com)
Lab Ruby walkthrough

- Break improper use of pseudo-random number generators to generate default passwords
  - Code uses Ruby to generate passwords
  - Seeds the random number generator with a constant
    - Random.new(seed)
  - Initial passwords are generated deterministically based on calls to RNG
  - One generated password (and the order in which it was generated) is known
    - Attack
      - Brute-force all seeds until a generated password matches your known password
      - Reveals the seed
      - Use position of known password to deduce password of first (root) user
Lab Ruby example

- Code to generate random usernames
- Find the seeds that produce “vwywbw” or “jozfbe” as random_name for the following code

```ruby
require 'digest'
s = Random.new(seed)
random_name = 6.times.map{('a'..'z').to_a[s.rand(('a'..'z').to_a.size)]}.join
```

repeat 6 times
to get 6 chars

create an array out of lowercase letters

generate size of character array to select from

generate random index into array of lowercase characters

join chars to form username of length 6
Lab Ruby example

- Find the seeds that produce “vwywbw” or “jozfbe” as the first username
- Invoke program as
  
  ruby InsecureCryptoStorage1.rb

```ruby
require 'digest'

seed=1
while seed < 20
  # Create new pseudo-random number generator with seed (loop tries the first 20)
  s = Random.new(seed)

  # Use PRNG to generate username
  # 6.times -> Generate 6 random characters
  # ('a'..'z').to_a -> Create array of lowercase letters
  # [s.rand(('a'..'z').to_a.size)] -> Index letter array with random number between 0,25
  random_name = 6.times.map{('a'..'z').to_a[s.rand(('a'..'z').to_a.size)]}.join

  print "Trying seed: ", seed, "\n"
  if (random_name == 'vwywbw') || (random_name == 'jozfbe')
    print "Found ", random_name, " as first userid for seed: ",seed,"\n"
    print "MD5 hash of ",random_name," is ",Digest::MD5.hexdigest(random_name),"\n"
    seed=seed+1
  else
    seed=seed+1
  end
end
```
Other helpful Ruby constructs

- Bounded ‘do’ loops

```ruby
10.times do |i|
  puts i
end
```

Before starting, do these two loops have the same output?

```ruby
s = Random.new(0)
10.times do |i|
  print i," ",s.rand(100),"\n"
end
```

```ruby
10.times do |i|
  s = Random.new(0)
  i.times{s.rand(100)}
  print i," ",s.rand(100),"\n"
end
```
Homework

- Insecure Cryptographic Storage Challenge #1
  - Reverse-engineer a simple rotation cipher

- Insecure Cryptographic Storage Challenge #2
  - Reverse-engineer a multi-alphabetic substitution cipher (Vigenere)
  - Use nodejs or Browser engine to execute JavaScript