Systems Security Group | Lancaster University

Security Fundamentals

Learning Objectives

- Revisit basic definitions used in security
- Learn the main security principles
- Learn about threats to security and ways to protect

Common information security targets

The classic top aspects of information security are the preservation of

- **Confidentiality:** ensuring that information is accessible only to those authorised to have access
- **Integrity:** safeguarding the accuracy and completeness of information and processing methods
- **Availability:** ensuring that authorised users have access to information and associated assets when required

Other definitions…

- **Anonymity/Untraceability** : Hiding identity
- **Pseudonymity** Acting through proxies?
- **Unlinkability** : Using the service multiple times without resource linking back to your identity
- **Copy protection, information flow control**
- **Data protection/personal data privacy**

Aspects of integrity and availability protection

- **Rollback** : Version control (going back to a working state in case of failure)
- **Authenticity**
- **Non-repudiation**
- **Audit**

Common questions regarding security

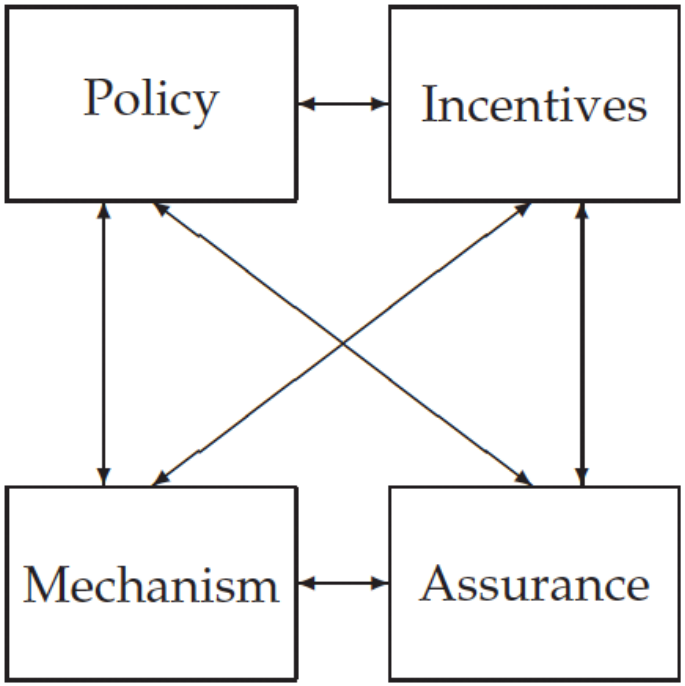
- Is my system secure?
- Factors that affect security
- What's required for an effective protection?

Security engineering

- *'… about building systems to remain dependable in the face of malice, error, or mischance'* [1]

- Focuses on tools, processes and methods
- Why?
  - Design, implement, test systems
- How easy is it?
  - Protect the wrong things…
  - Protect things in the wrong way…

A framework



Security Engineering Analysis Framework [1]

- **Policy:** What you are supposed to achieve
- **Mechanism:** What you assemble to implement the policy
- **Assurance:** Reliance you place on a mechanism
- **Incentive:** Motive for people or attackers to protect or attack a policy

What is a system?

'Simply stated, a system is an integrated composite of people, products, and processes that provide a capability to satisfy a stated need or objective.'

'Ignoring the human components, and thus neglecting usability issues, is one of the largest causes of security failure' [1]

Security principles

- Why do we need them?
- The security principles helps to achieve information security goals
  - Confidentiality, Integrity, Availability
- … most essential principles, regardless of the actual domain…

Simplicity

- *'Keep the design as simple and small as possible' [5]*
- It's easier the understand simple solutions
- Simple solution may be less likely to have vulnerabilities (compares to a complex solution)
- How about analysing and reviewing the system?

Open design

- *'The design should not be secret'* [5]
- The protection mechanism of a system shouldn't depend on secrecy
- Secrets may be hard to protect…

Compartmentalisation

- Organise resources into isolated groups or similar needs.
- Limit access to information based on tasks
- Have to identify similar needs
  - Object oriented programming

Least privilege

- *'Every program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job.'* – J. Saltzer

- Privileges should be reduced to the absolute minimum as long as users can complete their task

- Subjects of a system should not be granted access to objects other than those needed to complete their job

Trust and trustworthiness

- Trust Vs. Trustworthiness
  - A trusted system may misbehave and not meet the user's expectations
  - A trustworthy system satisfies the user's expectations
- Trust has to be minimised
- Trustworthiness has to be maximised

Fail-safe defaults

- A system should start in and return to a secure default state in case of failure
- A security mechanism can be enabled at system start-up and re-enabled whenever it fails
- It's an important principle in access control
  - Permission is denied unless explicitly granted
    - Whitelist approach

Complete mediation

- *'Every access to every object must be checked for authority'* [5]
- Access to any object must be monitored and controlled
- Ensure that access control mechanisms cannot be bypassed
- Protect sensitive information during transit/in storage, which requires data to be encrypted to achieve complete mediation

No single point of failure

- Build redundant security mechanisms
- Security should not rely on a single mechanism
- Prevent single points of failure
- Also known as defence in depth.

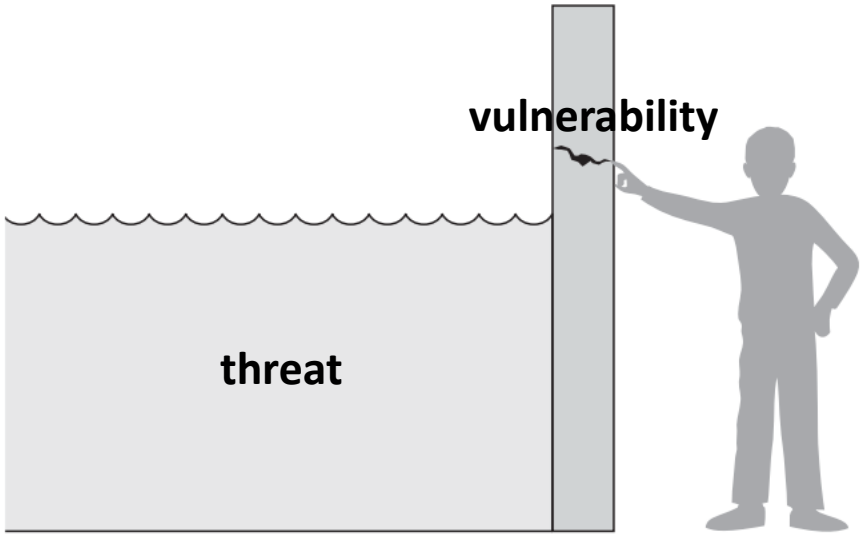
Usability

- Design usable security mechanisms
  - Security mechanisms should be easy to use
- Not only concerned with end users
  - System administrators, auditors, software engineers, etc.
- Security mechanisms should be designed with these users in mind

Threats to security and ways to protect

Vulnerabilities and threats [2]

- A **vulnerability** is a weakness in a system that might be exploited and cause loss or harm
- A **threat** to a system is a set of circumstances having the potential to cause loss or harm



Threats, controls and vulnerabilities[2]

Attacks and control

- The exploitation of a vulnerability perpetrates an **attack** on the system
- **Controls** are protective measures that could address problems

Threats Vs. Controls Vs. Vulnerabilities

'A *threat* is blocked by *control* of a *vulnerability*' [2]

Access controls under attack

- In reality most attacks take place on some type of access control → Where one can control who has access to what
- What makes it difficult for security professional is that there are several ways for a system to be attacked
- Before securing them, they should be identified

Security issues – Vulnerability analysis

- Look for security issues

- Carry out a vulnerability analysis
  - Look for holes that could be exploited
  - Carried out by scanning systems and identifying missing patches, misconfigured settings, programming code mistakes, etc.

What can go wrong…?

What can go wrong after running a vulnerability scanner and fixing all the issues?

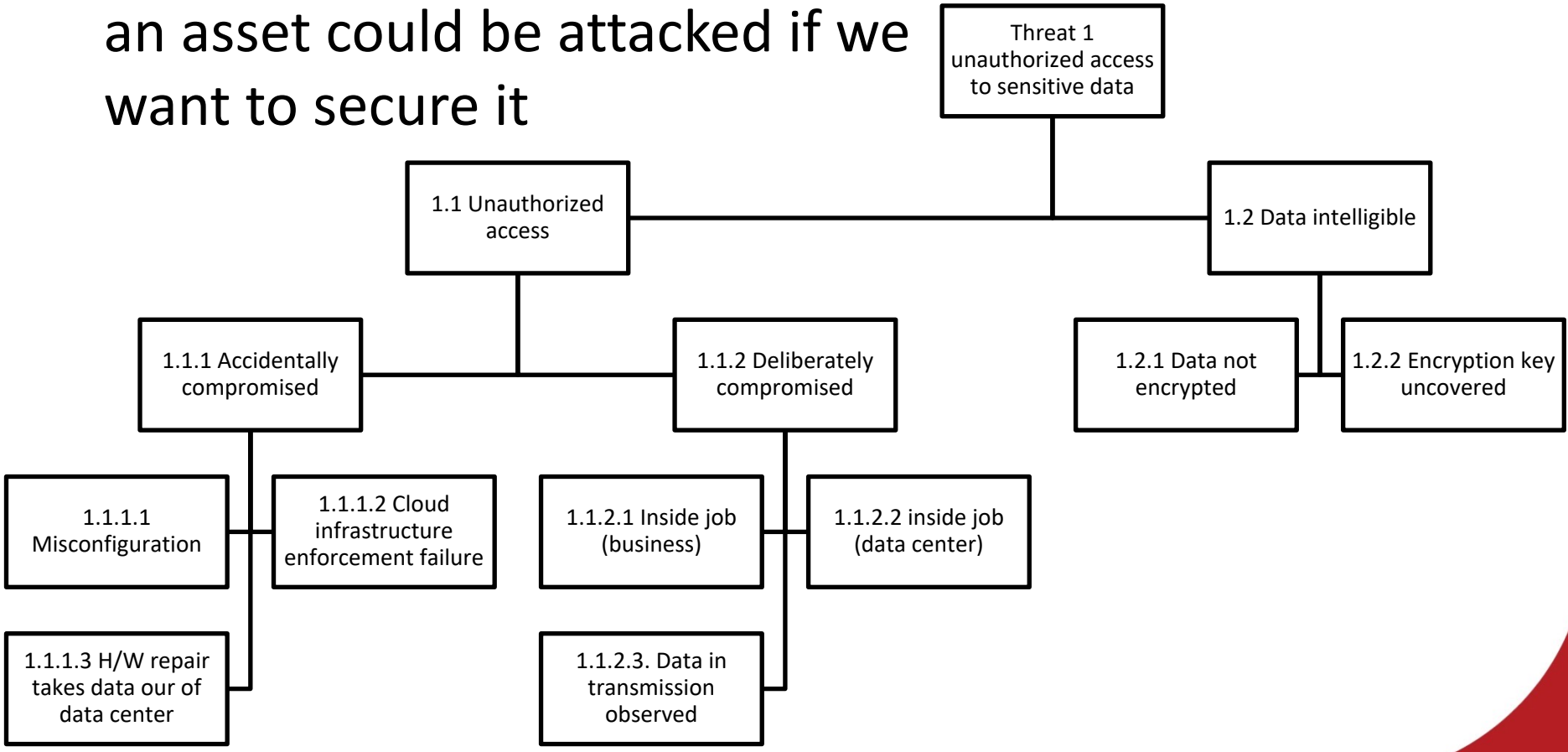
- How is this system connected to other systems?
- Are the sensitive data encrypted while in storage and transit?
- Who has access to this system?
- Can someone steal this system?
- Can someone insert a USB device and extract the data?
- What are the vectors that malware can be installed on the system?
- Is the system protected in the case of a disaster?
- Are there any access channels that are not auditable?

Threat modelling

- Threat modelling is a structured approach to identify potential threats that could exploit vulnerabilities
- Who would likely want to attack us?
- How could they successfully do this?
- Looks outward and try to figure out all the ways a structure could be attacked.

What has to be done?

Have to think about all the ways an asset could be attacked if we want to secure it

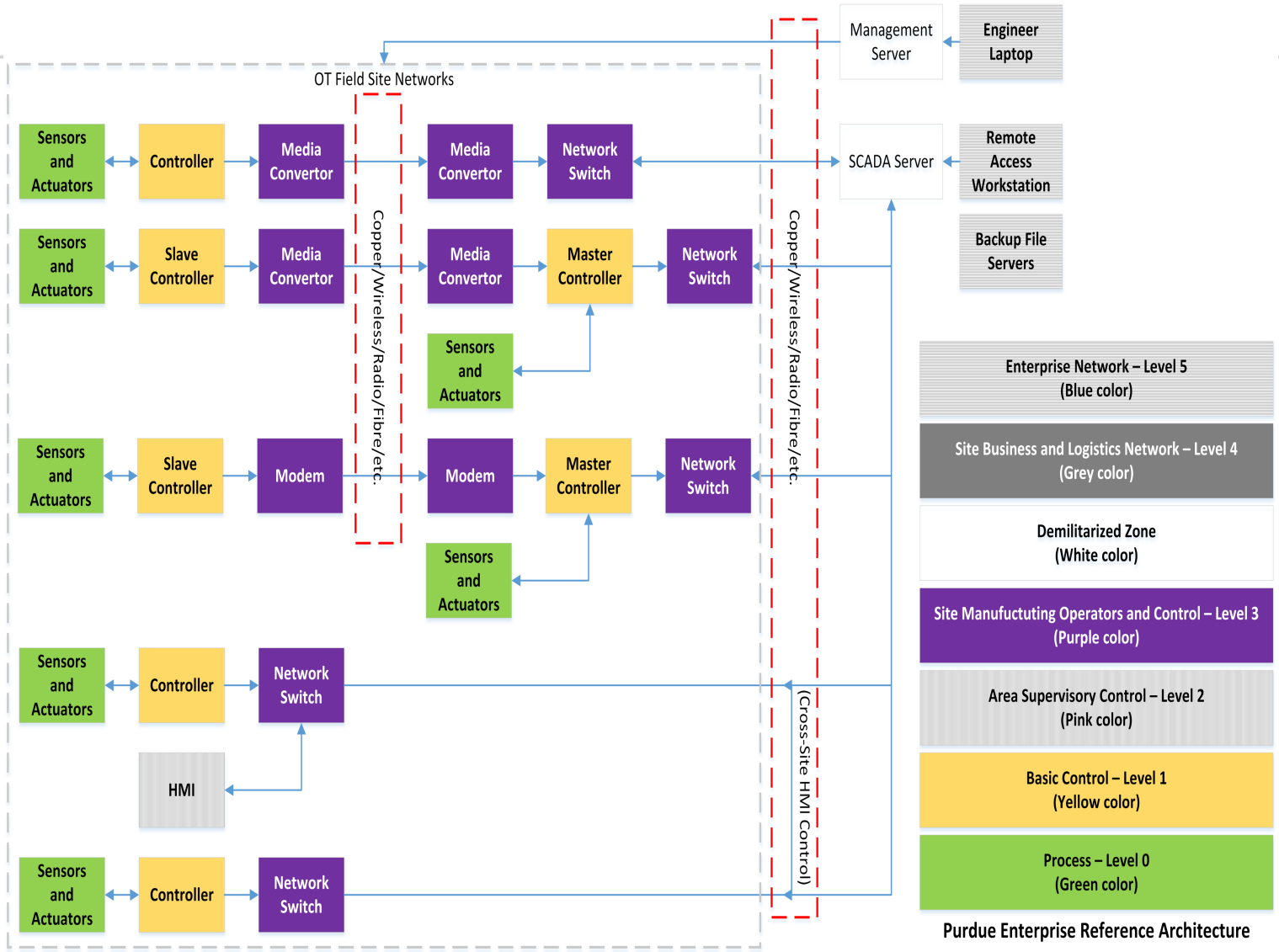


Security policies

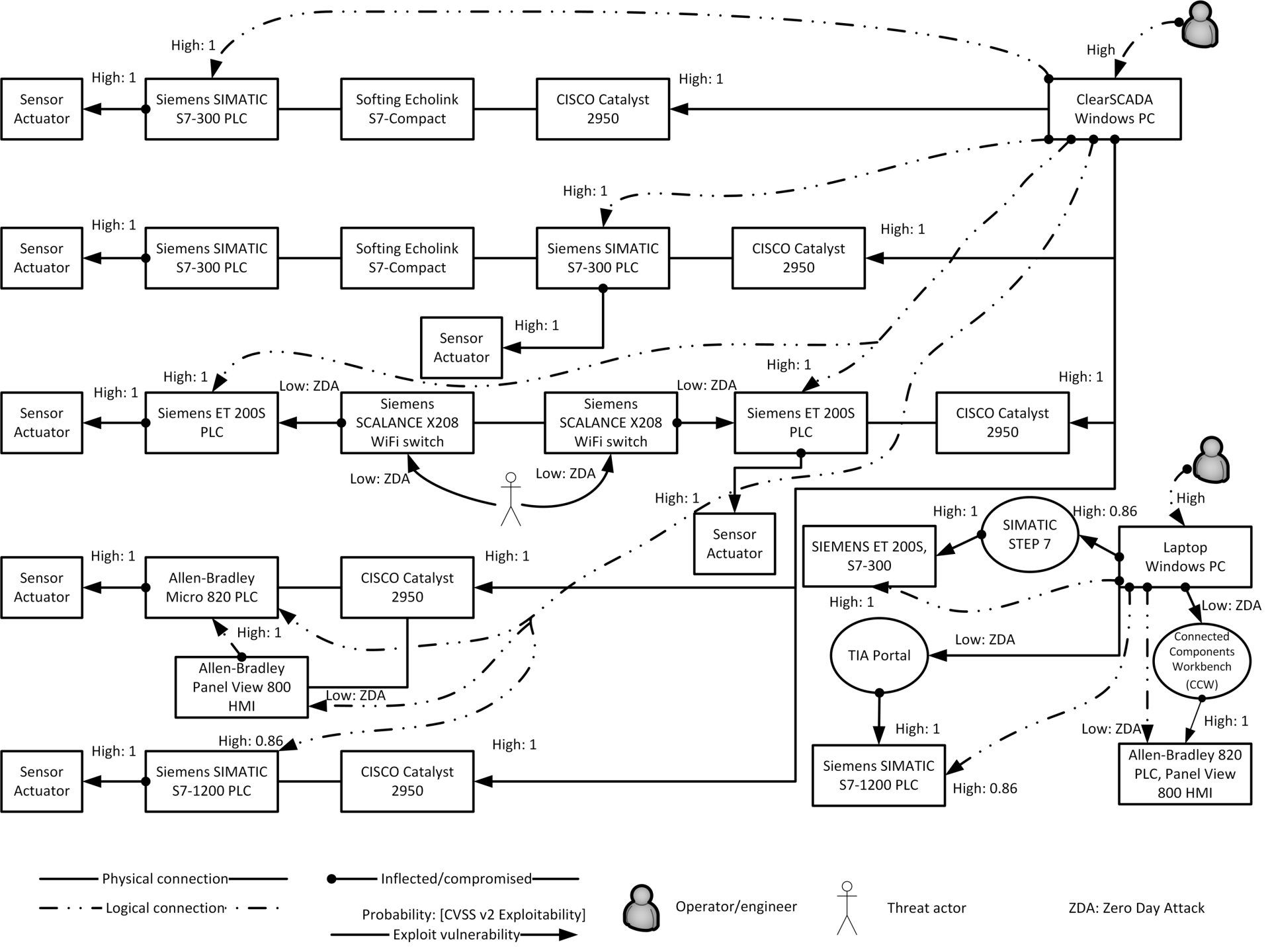
- Security requirement analysis
  - Identify assets and their values
  - Identify vulnerabilities, threats and risk priorities
  - Identify legal and contractual requirements
- Define security policies
- Security policy document
- Selection and implementation of controls

Markus Kuhn, Introduction to security,
https://www.cl.cam.ac.uk/teaching/0708/IntroSec/slides.pdf

Identify assets and values



Purdue Enterprise Reference Architecture


Sensor Actuator
Siemens SIMATIC S7-300 PLC
Softing Echolink S7-Compact
CISCO Catalyst 2950
ClearSCADA Windows PC
Siemens ET 200S PLC
Siemens SCALANCE X208 WiFi switch
Allen-Bradley Micro 820 PLC
Allen-Bradley Panel View 800 HMI
Siemens SIMATIC S7-1200 PLC
SIEMENS ET 200S, S7-300
SIMATIC STEP 7
Laptop Windows PC
TIA Portal
Connected Components Workbench (CCW)
Allen-Bradley 820 PLC, Panel View 800 HMI
High: 1
High: 0.86
Low: ZDA
High
Physical connection
Logical connection
Inflected/compromised
Probability: [CVSS v2 Exploitability]
Exploit vulnerability
Operator/engineer
Threat actor
ZDA: Zero Day Attack

Example: Risk identification and vulnerabilities

- Examples of threats on main assets
  - Radio jamming/data manipulation
  - Becoming a HMI
  - Backup server
  - …
- Examples of vulnerabilities
  - ClearSCADA server: CVE-2014-5411, CVE-2014-5412, CVE-2014-5413
  - Network switches: CVE-2001-0895, CVE-2014-5412
  - Controllers: Siemens SIMATIC S7-300 , S7-1200, ET 200S PLC, ...
  - Management server: SIMATIC STEP 7, Connecter Components Workbench, TIA Portal, ...

Gouglidis, Antonios and König, Sandra and Green, Benjamin and Rossegger, Karl and Hutchison, David (2018) *Protecting Water Utility Networks from Advanced Persistent Threats : A Case Study.* In: Game Theory for Security and Risk Management : From Theory to Practice. Static & Dynamic Game Theory: Foundations and Applications . Springer Birkhäuser, Basel.

Define suitable security policies

- The security requirements identified can be complex and may have to be abstracted first into high-level security policy

- A set of rules that clarifies which are and are not authorised, required, and prohibited activities, states and information flows

Markus Kuhn, Introduction to security,
https://www.cl.cam.ac.uk/teaching/0708/IntroSec/slides.pdf

Security policy document

- Understand what exactly security means for an organisation and what needs to be protected or enforced
- Document high-level security policies as a reference for anyone involved in implementing controls
- Lay out the overall objectives, principles, and the underlying threat model that are to guide the choice of mechanisms in the next step

Markus Kuhn, Introduction to security,
https://www.cl.cam.ac.uk/teaching/0708/IntroSec/slides.pdf

Selection and implementation of controls

- Issues addressed in a typical low-level organisation security policy
  - General (affecting everyone) and specific responsibilities for security
  - Name manager who ‘owns’ the overall policy an is in charge of its continued enforcement, maintenance, review, and evaluation of effectiveness
  - Name individual managers who ‘own’ individual information assets and are responsible for their day-to-day security
  - Reporting responsibilities for security incidents, vulnerabilities, software malfunctions

Markus Kuhn, Introduction to security,
https://www.cl.cam.ac.uk/teaching/0708/IntroSec/slides.pdf

… continued

- Mechanism for learning from incidents
- User training, documentation, and revision of procedures
- Physical security
  - Authorisation procedure for removal of property
  - Clear desk policy
  - Define security perimeter
  - …
- …

Markus Kuhn, Introduction to security,
https://www.cl.cam.ac.uk/teaching/0708/IntroSec/slides.pdf

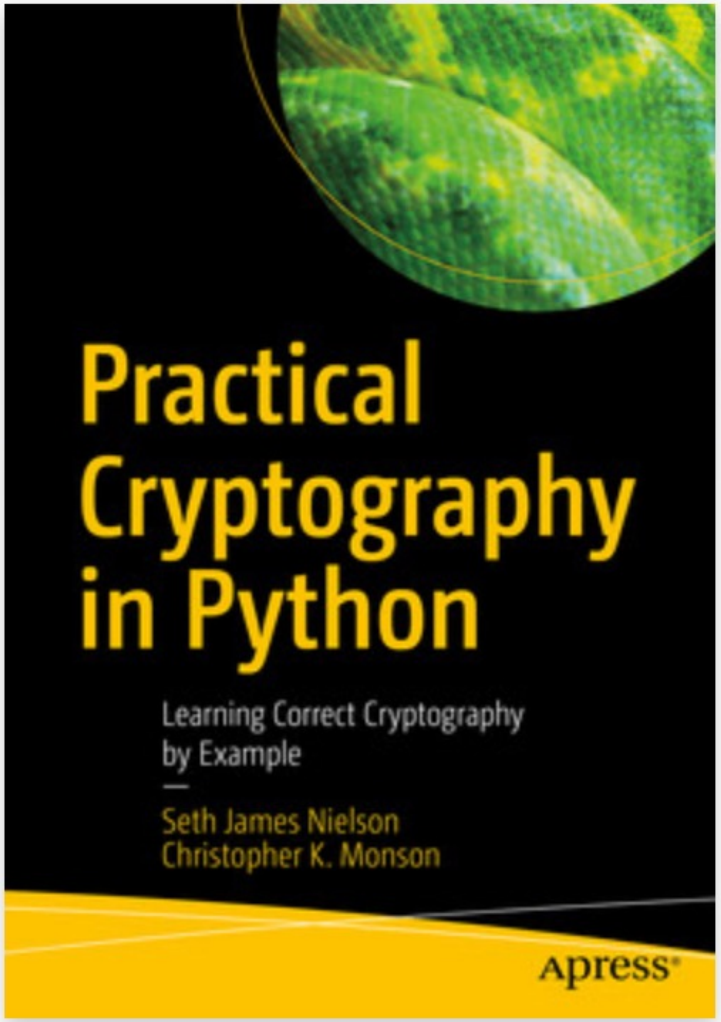
Questions?

References

[1] *Security Engineering: A Guide to Building Dependable Distributed Systems*, 2nd Edition, By Ross Anderson, Chapter ,1 https://www.cl.cam.ac.uk/~rja14/book.html
[2] *Security in Computing*, 5th Edition, By Charles P. Pfleeger, Shari Lawrence Pfleeger, Prentice Hall, Chapter 1, https://ptgmedia.pearsoncmg.com/images/9780134085043/samplepages/9780134085043.pdf
[3*] All In One – CISSP Exam Guide*, 5th Edition by Shon Harris. ISBN 978-0-07-160217-4
[4] *Systems Engineering Fundamentals*, DoD, January 2001, https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-885j-aircraft-systems-engineering-fall-2005/readings/sefguide_01_01.pdf
[5] The Protection of Information in Computer Systems, J. Saltzer and M. Schroeder, Proceedings of the IEEE, Vol. 63, No. 9, September 1975.
[6] William Stallings, Network and Internetwork Security: Principles and Practice. Englewood Cliffs, NJ: Prentice-Hall International, 1999.
[7] British Standard 7799 “Code of practice for information security management”
[8] German Information Security Agency’s “IT Baseline Protection Manual” http://www.bsi.bund.de/gshb/english/etc/
[9] US DoD National Computer Security Center Rainbow Series, for military policy guidelines http://www.radium.ncsc.mil/tpep/library/rainbow/

Week 11 Hashing

Recommended reading



The book is available to you via the library

Technology stack

- Python 3
  Link to a Python Cheat Sheet
- cryptography.io
  Link to the library

Topics

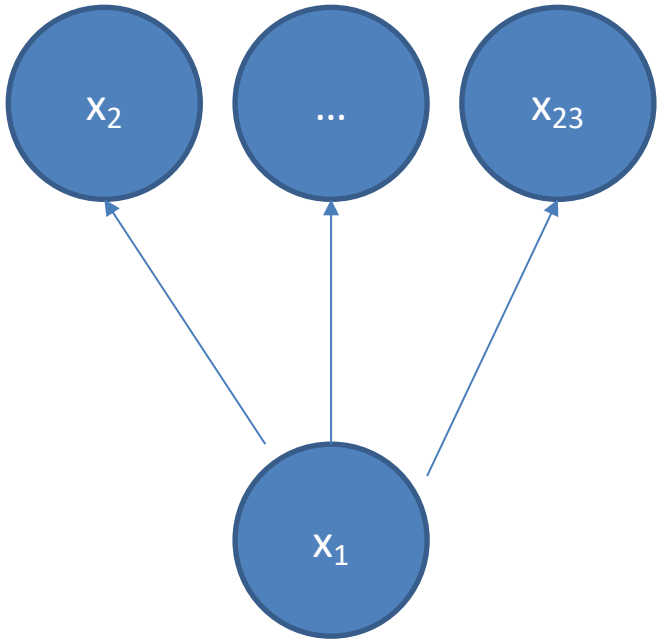
- Hashing & collisions
- How to create an avalanche calculator
- Other applications of hash functions

Recommended reading: Chapters 2 and 5 from the book of "Practical Cryptography in Python"

Reminder!

- A cryptographic hash function *H* must provide
  - Compression: e.g, *H*: {0,1}* → {0,1}^{160}
    finite hash, regardless of input
  - Efficiency: H: h(*x*) easy to compute for any *x*
  - One-way: given *y* it is infeasible to find *x*: h(*x*)=*y* (preimage resistance)
  - Weak collision resistance: for any given *x,* it should be difficult to find x' , *x'*≠*x* so that h(*x'*)=h(*x*) (2nd preimage resistance)
  - Strong collision resistance: it should be difficult to find any pair (*x, x'*) with *x*≠*x'* so that h(*x*)=h(*x'*) (collision resistance)
- Check message integrity!

Weak collision resistance



For any given *x,* it should be difficult to find x' , *x'*≠*x* so that h(*x'*)=h(*x*) (2nd preimage resistance)

Birthday attack example

A: x1 has the same b/d as x2, or ... or x23 (mutually exclusive)

$$P(A) = \underbrace{\frac{1}{365} + \ \ldots + \frac{1}{365}}_{32} \cong 0.063$$

Strong collision resistance

It should be difficult to find
any pair (*x, x'*) with $x \neq x'$ so that $h(x)=h(x')$
(collision resistance)

Birthday attack example

$$C(23,2) = \frac{23!}{21!\,2!} = \frac{21!\,22 * 23}{21!\,2} = 11 * 23 = 253$$

how many pairs from 23 ppl

A:2 people having b/d on a different day

$$P(A) = 1 - \frac{1}{365} = \frac{364}{365} \cong 0.99$$

B: All people having a different d/b

$$P(B) = P(A)^{253} \cong 0.49$$

C: At least one has the same b/d

$$P(C) = 1 - P(B) \cong 0.51$$

Hashing example

How to hash a string?

```
>>> print(hash1MD5.hexdigest())
'ed076287532e86365e841e92bfc50d8c'
>>> print(hash1MD5.digest())
b'\xed\x07b\x87S.\x866^\x84\x1e\x92\xbf\x
c5\r\x8c'
>>> len(hash1MD5.digest())
16
```

(16 x 8 = 128 bits long for MD5)

```
import hashlib
str1 = b"Hello World!"
hash1MD5 = hashlib.md5()
hash1MD5.update(str1)
hash2MD5 = hashlib.md5()
hash2MD5 = hashlib.md5(str1*100)
```

```
>>> print(hash2MD5)
c252ff6f54841f4970a9dd60aac5f5a2
>>> hash2MD5.digest_size
16
```

Size of binary values

```
>>> hash1MD5.update(b"Hello World1")

>>> bin1=bin(int (hash1MD5.hexdigest(),16))

>>> len(bin1)
129
```

whatever follows this is a binary representation
⇒ leads to 127 bits

```
>>> print(bin1)
0b10110001010100010100100001001101100101111110101001001111011001000011110010000101111111110100011111111011110000101000110111101
```

Collisions in MD5

Example of 2 different sequences of 128 byte that have to same MD5 hexdigest

```
d131dd02c5e6eec4693d9a0698aff95c2fcab58712467eab4004583eb8fb7f89
55ad340609f4b30283e488832571415a085125e8f7cdc99fd91dbdf280373c5b
d8823e3156348f5bae6dacd436c919c6dd53e2b487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080a80d1ec69821bcb6a8839396f9652b6ff72a70
```

```
d131dd02c5e6eec4693d9a0698aff95c2fcab50712467eab4004583eb8fb7f89
55ad340609f4b30283e4888325f1415a085125e8f7cdc99fd91dbd7280373c5b
d8823e3156348f5bae6dacd436c919c6dd53e23487da03fd02396306d248cda0
e99f33420f577ee8ce54b6708280d1ec69821bcb6a8839396f965ab6ff72a70
```

Source: https://www.mscs.dal.ca/~selinger/md5collision/
Paper: X. wang, H. Y, “How to Break MD5 and Other Hash Functions”, http://merlot.usc.edu/csac-f06/papers/Wang05a.pdf

How to…

```
>>> fout = open('bin1', 'wb')

>>> data1 =
b"\xd1\x31\xdd\x02\xc5\xe6\xee\xc4\x69\x3d\x9a\x06\x98\xaf
\xf9\x5c\x2f\xca\xb5\x87\x12\x46\x7e\xab\x40\x04\x58\x3e\x
b8\xfb\x7f\x89\x55\xad\x34\x06\x09\xf4\xb3\x02\x83\xe4\x88
\x83\x25\x71\x41\x5a\x08\x51\x25\xe8\xf7\xcd\xc9\x9f\xd9\x
1d\xbd\xf2\x80\x37\x3c\x5b\xd8\x82\x3e\x31\x56\x34\x8f\x5b
\xae\x6d\xac\xd4\x36\xc9\x19\xc6\xdd\x53\xe2\xb4\x87\xda\x
03\xfd\x02\x39\x63\x06\xd2\x48\xcd\xa0\xe9\x9f\x33\x42\x0f
\x57\x7e\xe8\xce\x54\xb6\x70\x80\xa8\x0d\x1e\xc6\x98\x21\x
bc\xb6\xa8\x83\x93\x96\xf9\x65\x2b\x6f\xf7\x2a\x70"

>>> fout.write(data1)
>>> fout.close()
```

Calculating the difference of outputted bits between two hexidecimal value hashes

Systems Security Group | Lancaster University

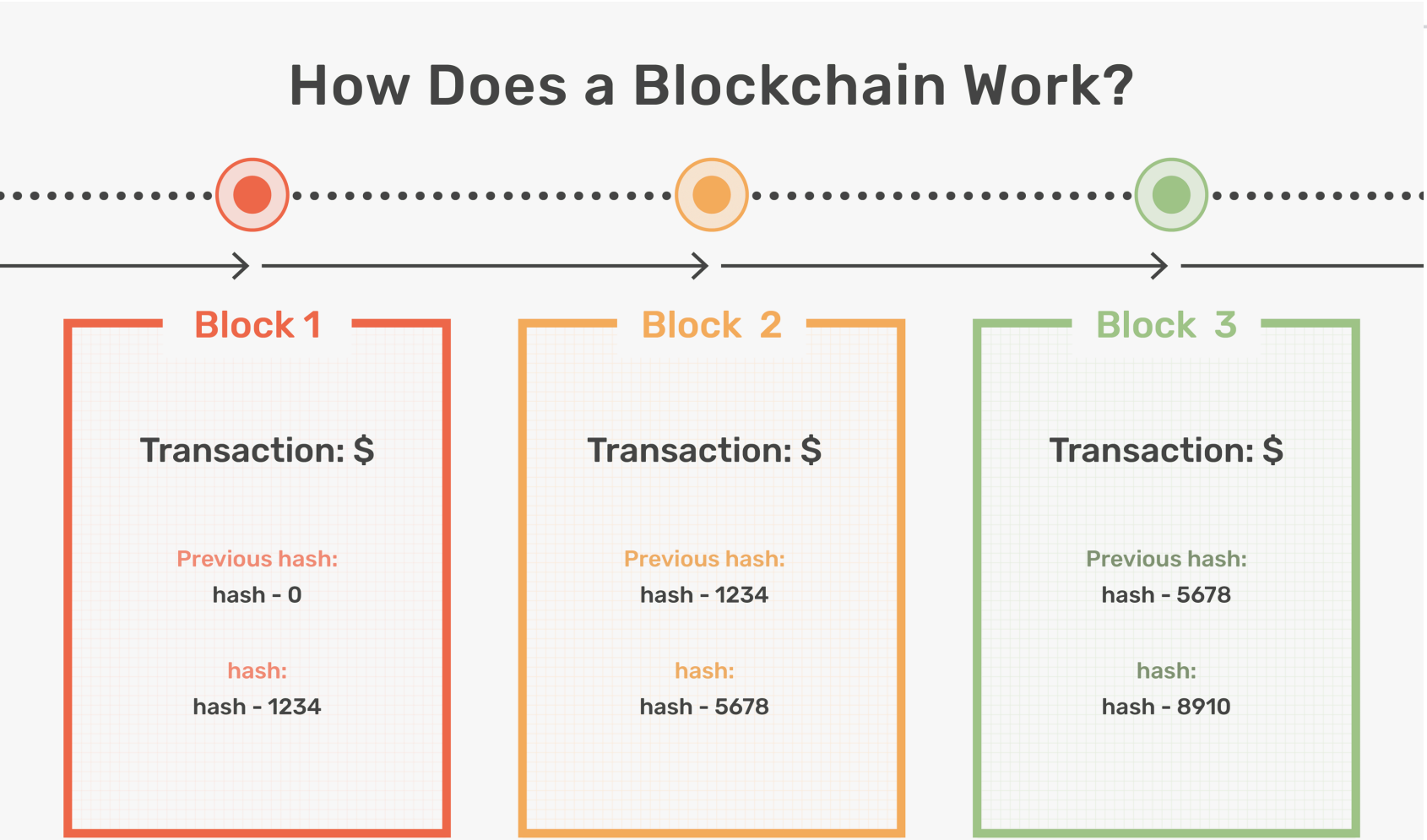
Avalanche effect

Example from "Practical Cryptography in Python"

```
MD5(bob):
   9   f   9   d   5   1   b   c   7   0   e   f   2   1   c   a
10011111100111010101000110111100011100001110111100100001110010 10
   5   c   1   4   f   3   0   7   9   8   0   a   2   9   d   8
01011100000101001111001100000111100110000000101000101001110110 00

MD5(cob):
   3   8   6   6   8   5   f   0   6   b   e   e   c   b   9   f
00111000011001101000010111110000011010111110111011001011100111 11
   3   5   d   b   2   e   2   2   d   a   4   2   9   e   c   9
00110101110110110010111000100010110110100100001010011110110010 01

Changed Bits:

X_X__XXXXXXXX_XXXX_X_X___X__XX_____XX_XX_______XXXX_X_X__X_X_X_X
_XX_X__XXX__XXXXXX_XXX_X__X__X_X_X____X__X__X___X_XX_XXX___X___X
```

- Difference is 64 bits
- Avalanche helps collision resistance

Application of hash functions



the balance https://tinyurl.com/4b72z94u

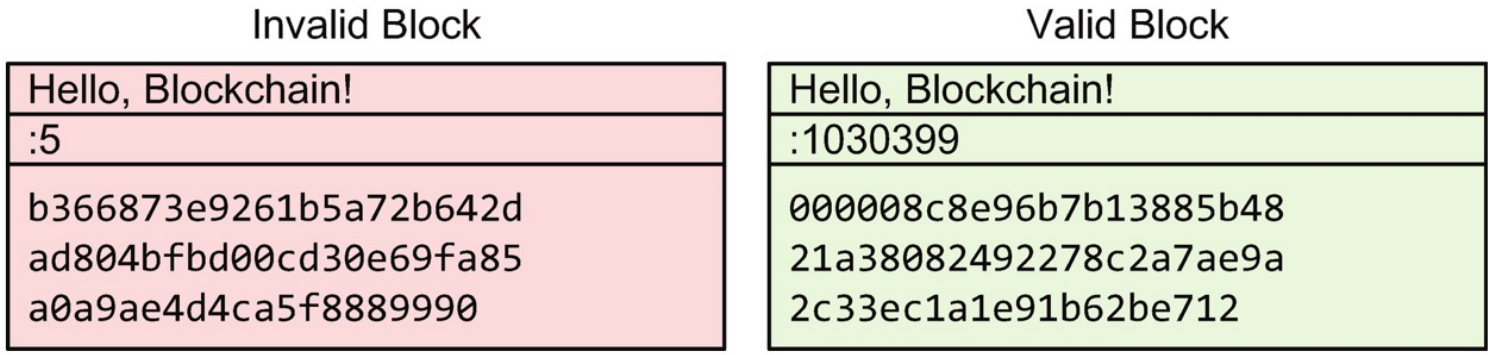
Application of hash functions (2)

prevents users going back blocks and altering them since they'll be detected

- Main idea
  - Protect each block with a hash
- Incentive
  - Give an award when a new block is added, but make it difficult to produce it.
- Sequence of actions
  - A user can request a transaction
  - Miners get the request and create a candidate block
  - The block has the transactions, metadata, etc.
  - It's added on the blockchain when the miner solver a puzzle!

What's the puzzle?

- Find a SHA-256 hash value that is smaller then a threshold.
- The threshold defined the difficulty of the network
- The puzzle is solved when adding a nonce to the block will result producing a hash value having a certain number of leading zeros.



Source: Chapter 2: Applied Cryptography in Python

More applications… Hash-based Message Authentication Codes (HMAC)

- Collision resistance + unforgeability

```
from cryptography.hazmat.primitives import hashes, hmac
import os
key = os.urandom(32)

h_sender = hmac.HMAC(key, hashes.SHA256())
h_sender.update(b"This is my message")
signature = h_sender.finalize()
```

Sender-side code

```
h_receiver = hmac.HMAC(key, hashes.SHA256())
h_receiver.update(b"This is my message")
try:
        h_receiver.verify(signature)
        print(b"OK")
except:
        print(b"Something went wrong")
```

Receiver verification

code w/out fails try/except

Structure of your code…

Modules you want to import

```
import XYZ
```

List of functions you implement

```
def  myFunction():
     # TODO

     return # TODO
```

Have a main section to call your functions

```
if __name__ == "__main__":
     x = myFunction()
```