Air Force Sustainment Center

Uncomfortable Truths About Cybersecurity

Lynn Wallace, CSSLP, MS SysEng
lynn.wallace@us.af.mil
dsn 775-3964

PURPOSE

Cybersecurity

Software Engineering

4 - EVER!

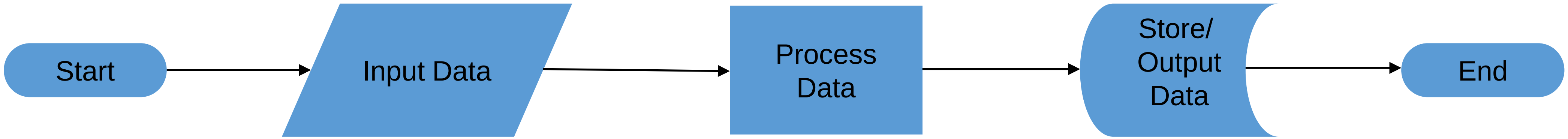
Overview

- **Uncomfortable truths about software and the people who write it.**
- **We don’t talk anymore.**
- **“We have an air gap.”** 😆
- **DevSecOps is great. It’s also not enough!**
- **Fixes.**

Uncomfortable Truths About Software

and the People Who Write It

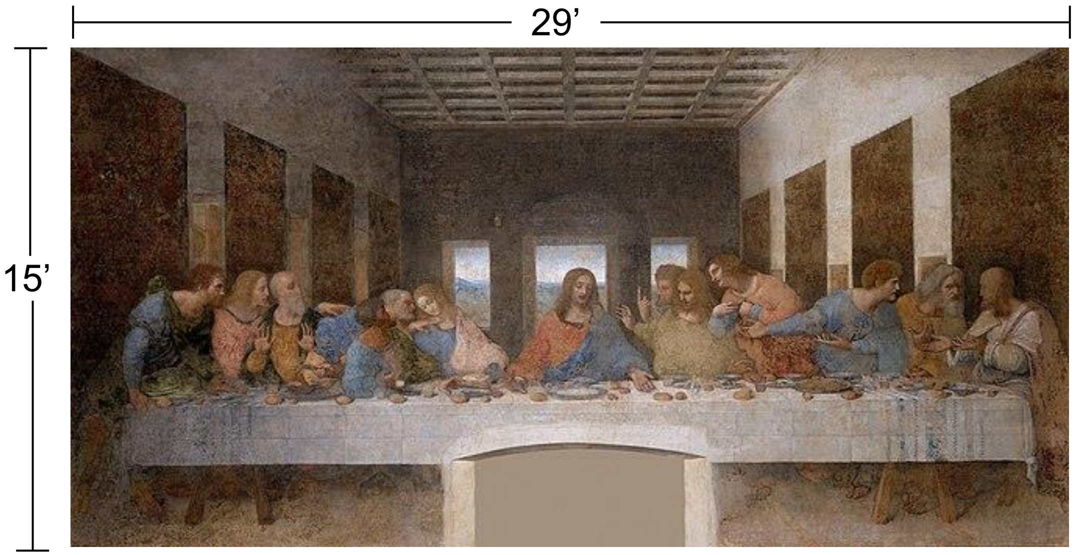
The Nature of Software



- **Software automates manual* processes**
- **Software processes data**
  - **Data has value – to someone – some more than others**
- **Insubstantial, Malleable, Complex**

* Of course, computers work a lot faster, can access signals, etc. But most code is stepped through, desk checked, or simulated during development.

Measuring Software



3 years

16 years

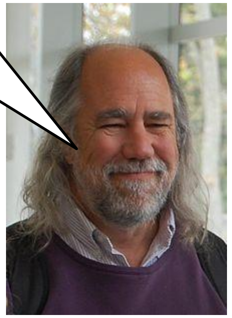
Source Lines of Code

- **Counting SLOC in software is like counting brush strokes in art.**
- **Good software estimation is accurate within 25%, 75% of the time.**
- **SLOC-based coverage is ~~awful~~ the most useful measure we have.**

The Nature of Software

- Perfect code doesn't exist (CompSci 101, Lesson 1)
  - The only risk-free software is what you're *not* running
- Keep it simple, stupid (KISS, CompSci 101, Lesson 2)
- Software is never done
  - Everything that happens in acquisition also happens in sustainment
  - Somewhere in your enterprise, there is software at every stage of its lifecycle
- Literally anyone can write software that literally anyone can use
  - 50-year-old worldwide supply chain
  - Few know all the software they're running
  - Nobody* knows who wrote their software

Old software doesn't die. You have to kill it.

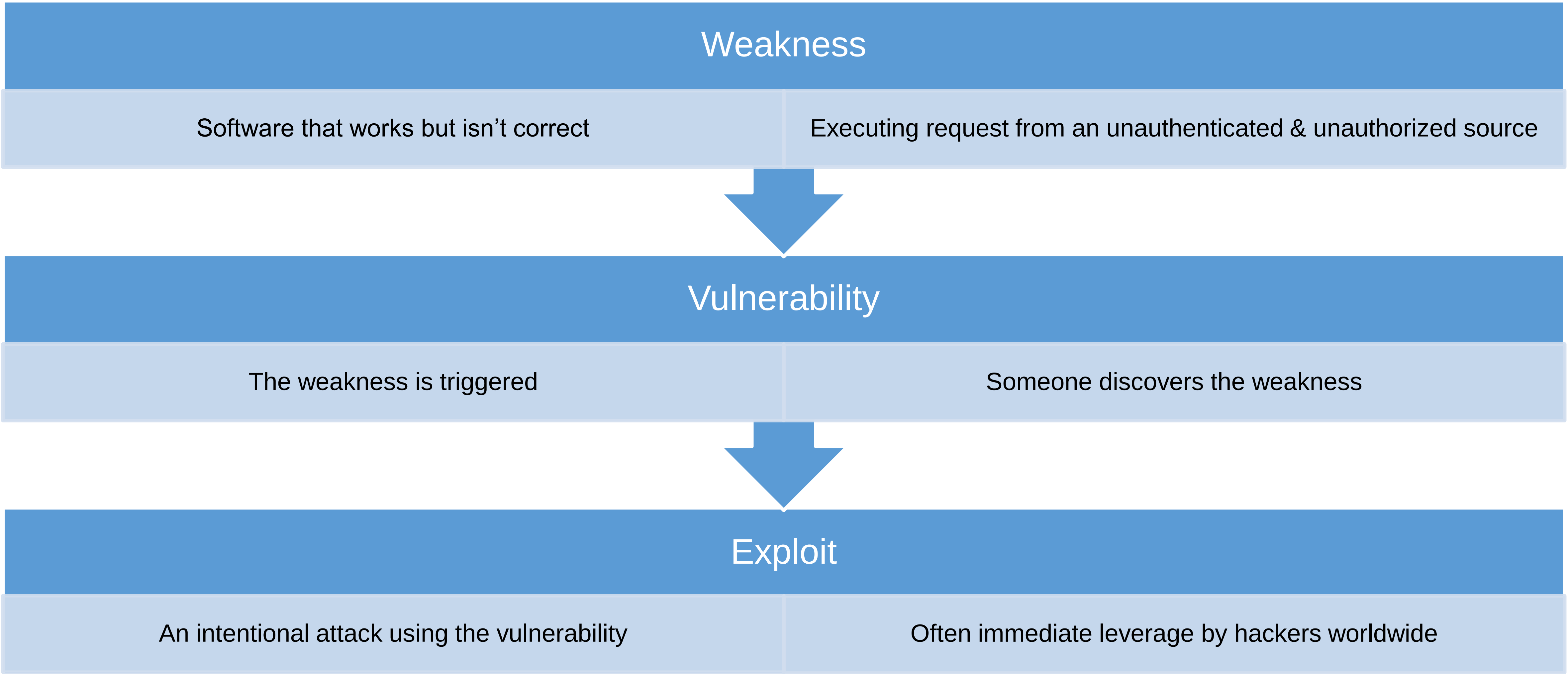


Grady Booch

About Programmers

- **Programmers learn how to program**
  - **They write code that *does things,* not *is things***
  - **“Quality” and “Security” are poorly defined**
  - **Programmers don’t “do” security**
  - **They *don’t know* security (or safety, reliability, human factors, management…)**
- **Every new program is a novel problem**
- **Good programmers are:**
  - **Curious lifelong learners**
  - **Ingenious outside-the-box problem solvers**
  - **Arrogant**
  - **Lazy-ish: reuse & whatever works**

What Programmers Don’t Know



Weakness

Software that works but isn’t correct

Executing request from an unauthenticated & unauthorized source

Vulnerability

The weakness is triggered

Someone discovers the weakness

Exploit

An intentional attack using the vulnerability

Often immediate leverage by hackers worldwide

Pop Quiz for Programmers

1. **If you create a vulnerability in your software, who did it?**

   (0 points): ____

Pop Quiz for Programmers

1. **If you create a vulnerability in the software, who did it?**

   (0 points): You (duh) ______________________________

2. **Who should fix it?**

   (1 point): ____________________________________

Pop Quiz for Programmers

1. **If you create a vulnerability in the software, who did it?**

   (0 points): You (duh)

2. **Who should fix it?**

   (1 point): You (duh)

3. **Who *else* can fix it?**

   (5 points):

Pop Quiz for Programmers

1. **If you create a vulnerability in the software, who did it?**

   (0 points): You (duh)

2. **Who should fix it?**

   (1 point): You (duh)

3. **Who *else* can fix it?**

   (5 points): Your colleagues

4. **Who *can't* fix it?**

   (20 points):

Pop Quiz for Programmers

1. **If you create a vulnerability in the software, who did it?**

   (0 points): You (duh)

2. **Who should fix it?**

   (1 point): You (duh)

3. **Who *else* can fix it?**

   (5 points): Your colleagues

4. **Who *can't* fix it?**

   (20 points): Cybersecurity professionals

5. **Extra credit: Why would you create a vulnerability?**

   (All the points):

Pop Quiz for Programmers

1. **If you create a vulnerability in the software, who did it?**

   (0 points): You (duh)

2. **Who should fix it?**

   (1 point): You (duh)

3. **Who *else* can fix it?**

   (5 points): Your colleagues

4. **Who *can't* fix it?**

   (20 points): Cybersecurity professionals

5. **Extra credit: Why would you create a vulnerability?**

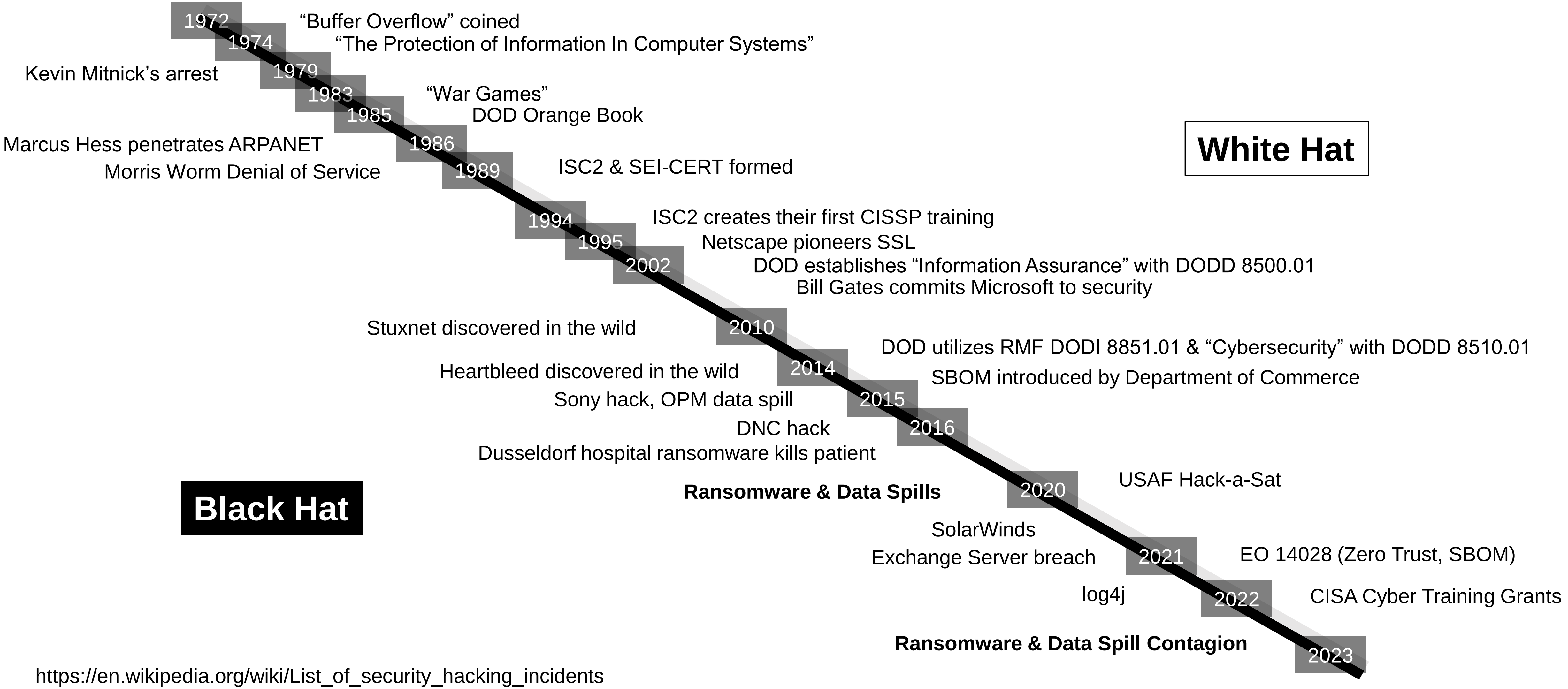
   (All the points): Ignorance, laziness, accident, malice, honest oversight, insecure tools

More of What Programmers Don’t Know

Taught in school	➔➔➔	Learned in career	➔➔➔	CSSLP certification
Economy of mechanism (KISS)				
		Fail-safe defaults		
				Complete mediation
Open design				
		Separation of duties		
	Least privilege			
				Least common mechanism
			Psychological acceptability	
		Work factor		
				Compromise recording
	Component reuse			
				Resiliency
			Defense in depth	

We don't talk anymore

The History of Cybersecurity



1972 “Buffer Overflow” coined

1974 “The Protection of Information In Computer Systems”

Kevin Mitnick’s arrest 1979

1983 “War Games”

1985 DOD Orange Book

Marcus Hess penetrates ARPANET 1986

Morris Worm Denial of Service 1989 ISC2 & SEI-CERT formed

1994 ISC2 creates their first CISSP training

1995 Netscape pioneers SSL

2002 DOD establishes “Information Assurance” with DODD 8500.01

Bill Gates commits Microsoft to security

Stuxnet discovered in the wild 2010

2014 DOD utilizes RMF DODI 8851.01 & “Cybersecurity” with DODD 8510.01

Heartbleed discovered in the wild

2015 SBOM introduced by Department of Commerce

Sony hack, OPM data spill

DNC hack 2016

Dusseldorf hospital ransomware kills patient

Ransomware & Data Spills 2020 USAF Hack-a-Sat

SolarWinds

Exchange Server breach 2021 EO 14028 (Zero Trust, SBOM)

log4j 2022 CISA Cyber Training Grants

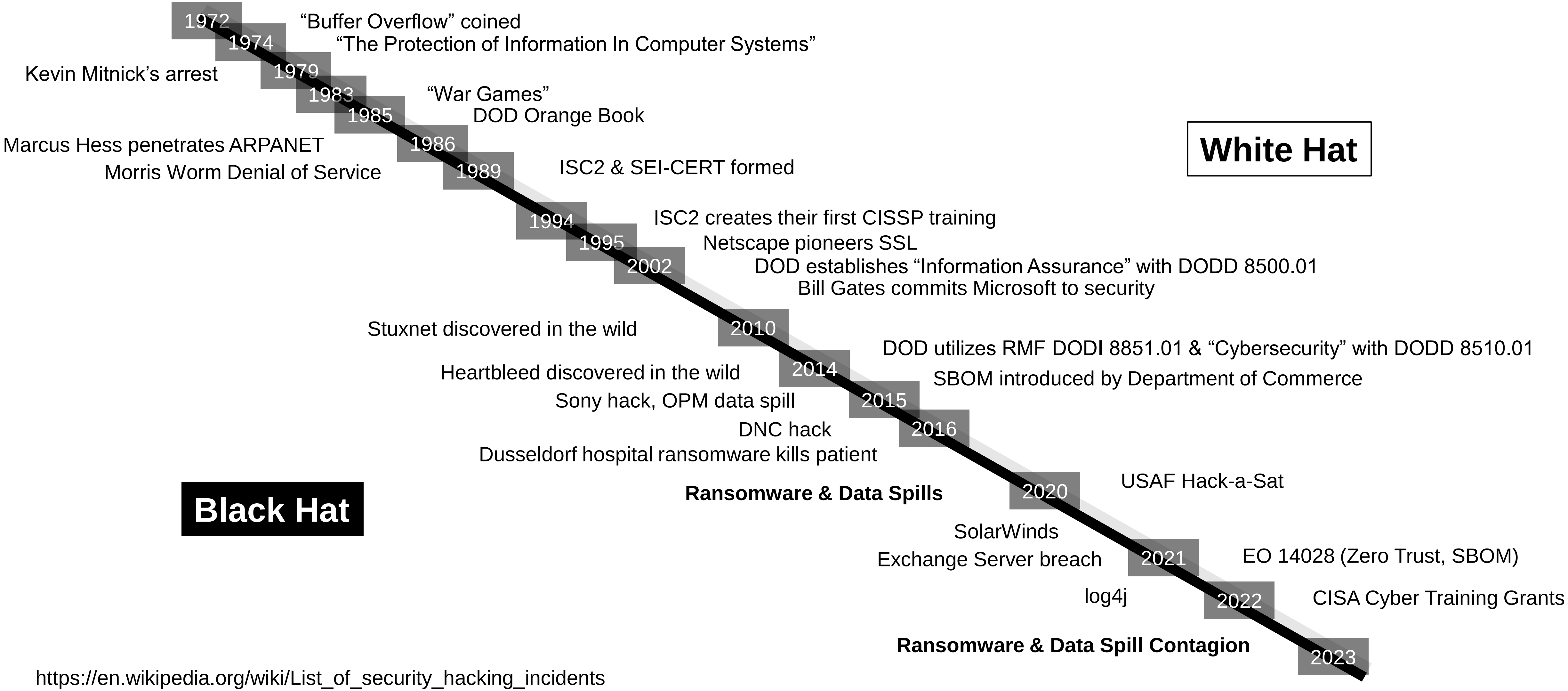
Ransomware & Data Spill Contagion 2023

White Hat

Black Hat

https://en.wikipedia.org/wiki/List_of_security_hacking_incidents

The History of Computer Science



1972 “Buffer Overflow” coined

1974 “The Protection of Information In Computer Systems”

Kevin Mitnick’s arrest 1979

1983 “War Games”

1985 DOD Orange Book

Marcus Hess penetrates ARPANET 1986

Morris Worm Denial of Service 1989 ISC2 & SEI-CERT formed

White Hat

1994 ISC2 creates their first CISSP training

1995 Netscape pioneers SSL

2002 DOD establishes “Information Assurance” with DODD 8500.01

Bill Gates commits Microsoft to security

Stuxnet discovered in the wild 2010

DOD utilizes RMF DODI 8851.01 & “Cybersecurity” with DODD 8510.01

Heartbleed discovered in the wild 2014

SBOM introduced by Department of Commerce

Sony hack, OPM data spill 2015

DNC hack 2016

Dusseldorf hospital ransomware kills patient

Black Hat

Ransomware & Data Spills 2020 USAF Hack-a-Sat

SolarWinds

Exchange Server breach 2021 EO 14028 (Zero Trust, SBOM)

log4j 2022 CISA Cyber Training Grants

Ransomware & Data Spill Contagion 2023

https://en.wikipedia.org/wiki/List_of_security_hacking_incidents

What We Don’t Know

Programmers

- Computer Science = Cybersecurity
- Security is fundamental to software
- Security is *our* job before it’s anyone else’s
- Information has inherent and imputed value
- How hackers work
- We *must* start thinking like an attacker
- How *secure* software is developed
- How easy secure development is

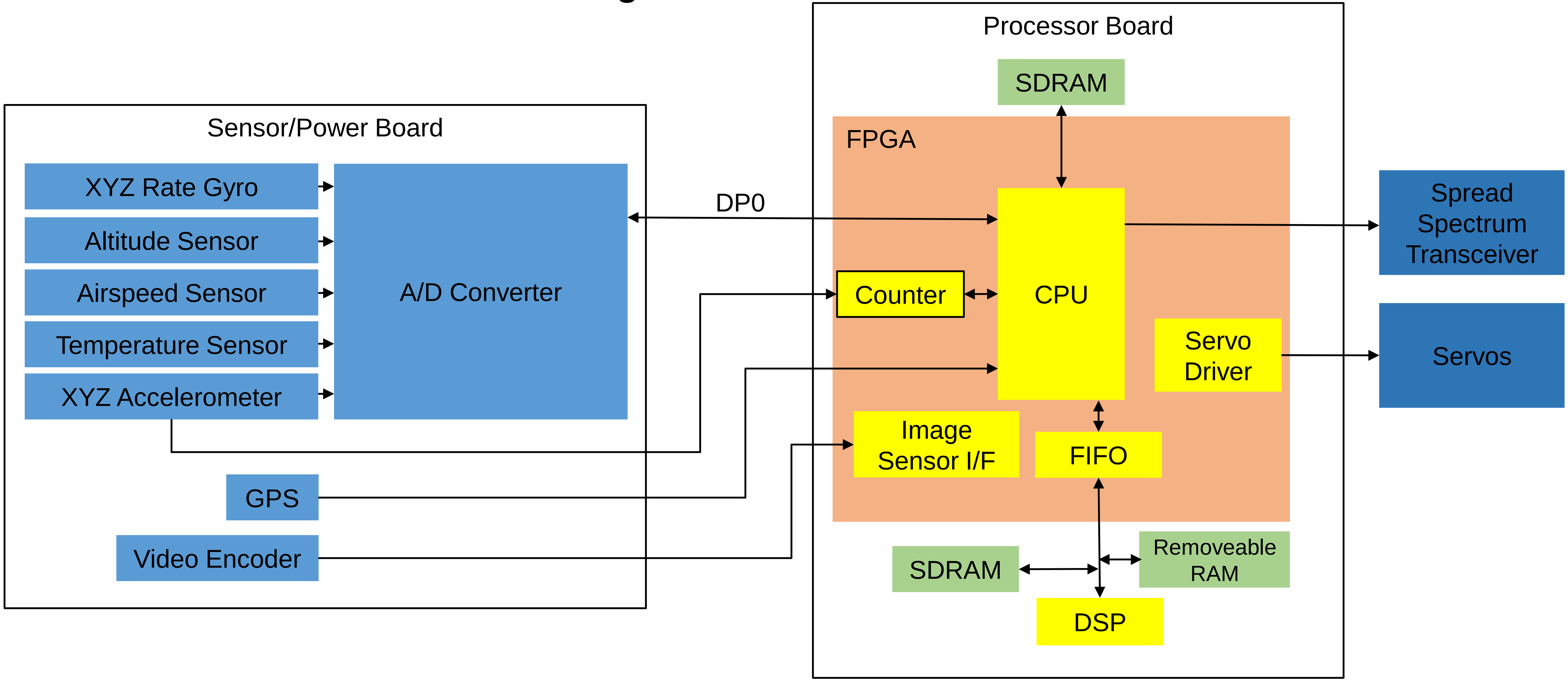
Cybersecurity Professionals

- How programmers think
- How programmers work
- How software is created
- How *secure* software is developed
- How working at the system level doesn’t work at the software level

"We have an air gap"

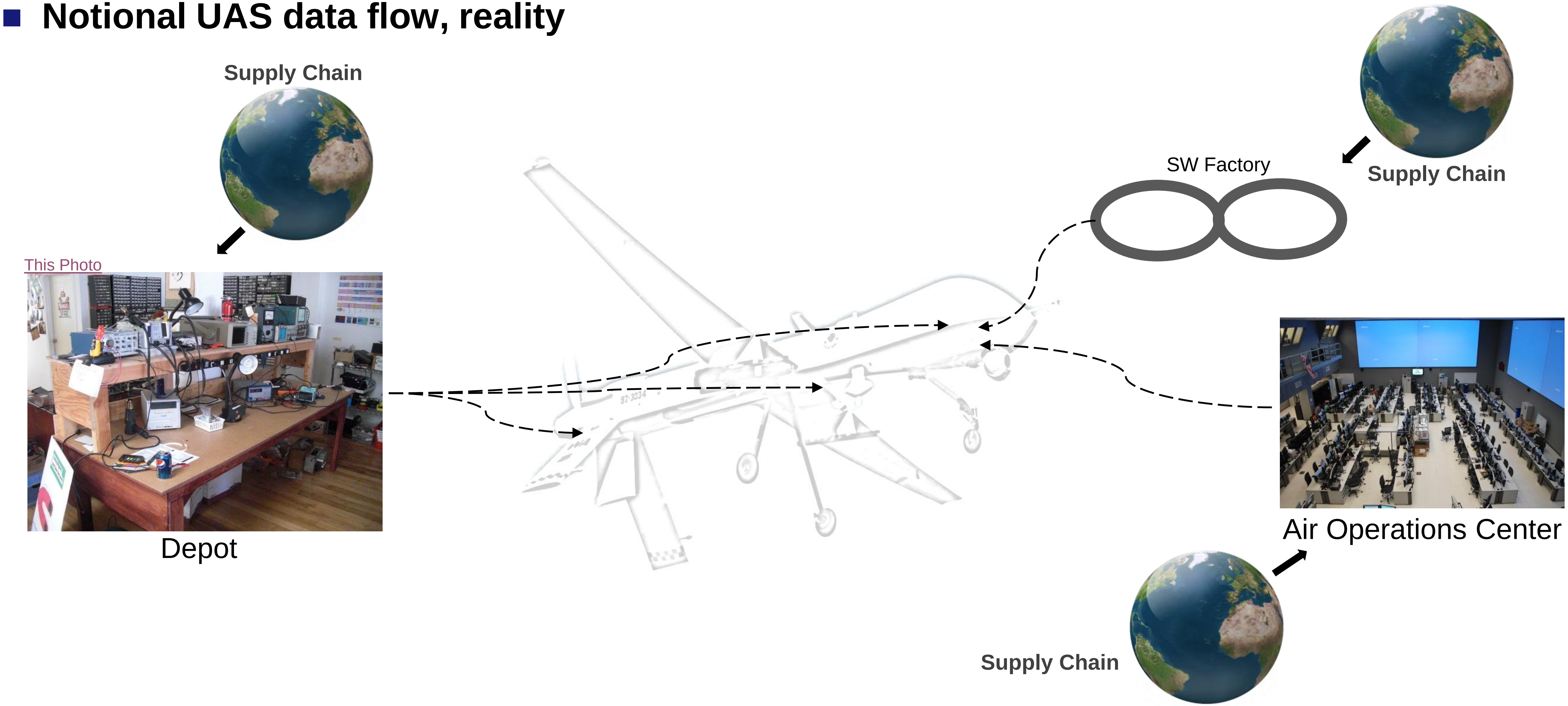
Air Gap = 1980's Network

- **Notional UAS data flow as designed**

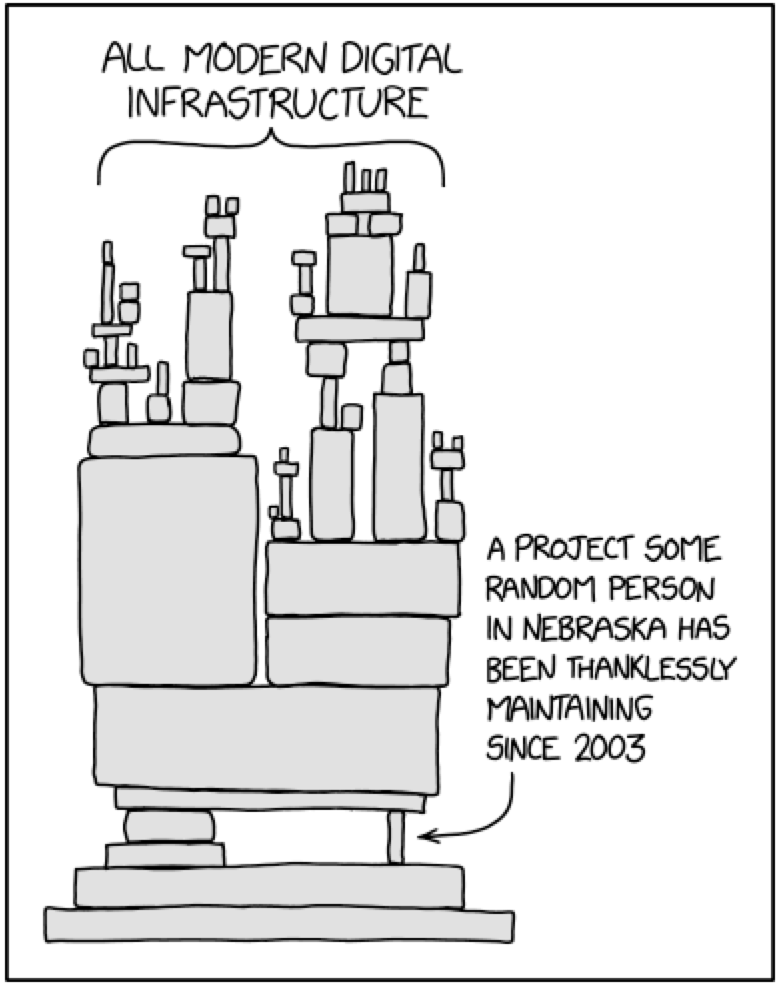


Air Gap = 1980's Network

- **Notional UAS data flow, reality**

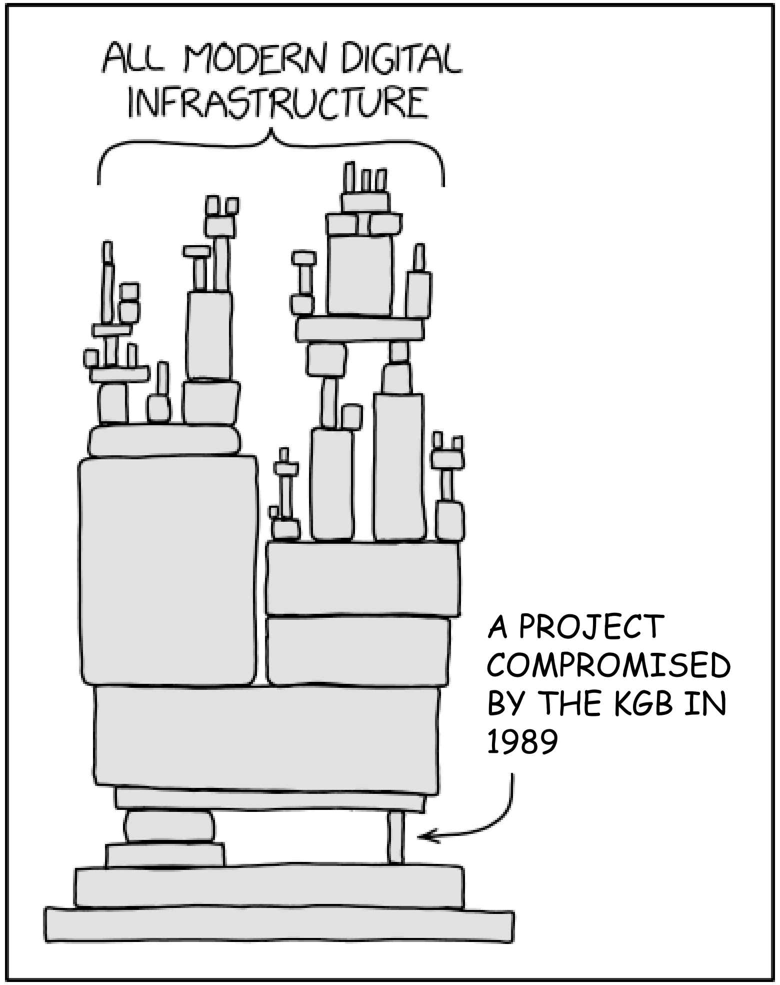


50-Year-Old Supply Chain


ALL MODERN DIGITAL INFRASTRUCTURE
A PROJECT SOME RANDOM PERSON IN NEBRASKA HAS BEEN THANKLESSLY MAINTAINING SINCE 2003

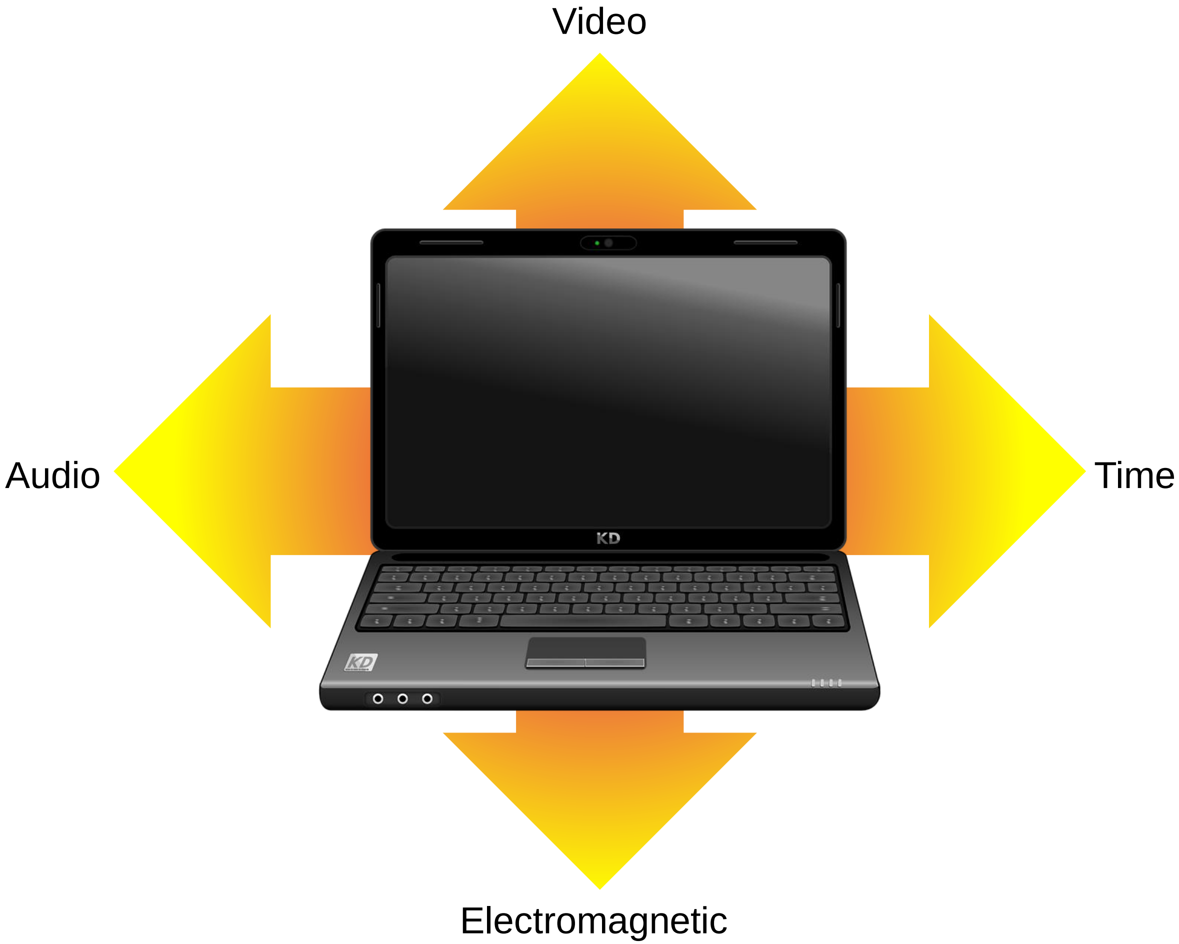
https://xkcd.com/2347

50-Year-Old Supply Chain


ALL MODERN DIGITAL INFRASTRUCTURE
A PROJECT COMPROMISED BY THE KGB IN 1989

Side Channels & Insider Threats

- **Side Channel: Using Laws of Physics to Bridge Air Gaps**
  - **Electromagnetic Radiation**
  - **Sound**
  - **Light**
  - **Execution Time**
- **Insider Threats**
  - **Nefarious**
  - **Clumsy**
  - **Deceived**
  - ***Includes the whole software supply chain***



DevSecOps is great!

It’s also not enough!

Log4j, December 2021

- **Very common Java logging service that permits attacker code execution on the hosting server:**
- **Amazon Web Services, Cloudflare, iCloud, many others**
- **40% remain unpatched (Feb. 23)**

  (https://securityintelligence.com/articles/log4j-downloads-vulnerable/)

- **CISA's investigation report:**
  - https://www.cisa.gov/news-events/news/apache-log4j-vulnerability-guidance

Designed-in Vulnerabilities

Security Analytics: >70% of exploited vulnerabilities are *design flaws*

Other Recent Examples:

- **Heartbleed, 2014**

  Heartbeat feature with design flaw that exposed server memory to attacker

- **Equifax, 2017**

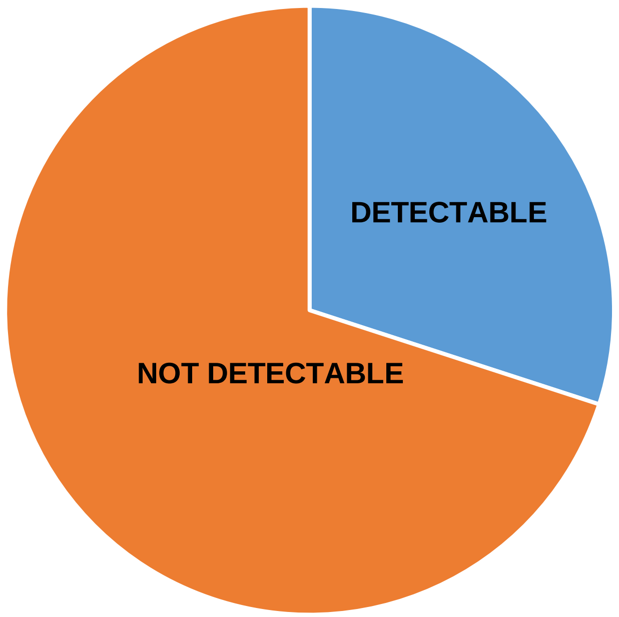
  XML External Entity (XXE)

  Inadequate input validation

- **Cambridge Analytica, 2018**

  Inadequate access control

Top 25 CWE Detectable in Source Code



Briefer: Lynn Wallace (517 SWES/MXDPA)

Designed-in Vulnerabilities

Common Weakness Enumeration Top 25 (out of 933 total) : https://cwe.mitre.org/top25/

Difficult or impossible to detect in source code

1. Out-of-Bounds Write
2. Improper Neutralization of Input During Web Page Generation ("Cross-site Scripting")
3. Improper Neutralization of Special Elements used in an SQL Command ("SQL Injection")
4. Use After Free
5. Improper Neutralization of Special Elements used in an OS Command ("OS Command Injection")
6. Improper Input Validation
7. Out-of-Bounds Read
8. Improper Limitation of a Pathname to a Restricted Directory ("Path Traversal")
9. Cross-Site Request Forgery (CSRF)
10. Unrestricted Upload of File With Dangerous Type
11. Missing Authorization
12. NULL Pointer Dereference
13. Improper Authentication
14. Integer Overflow or Wraparound
15. Deserialization of Untrusted Data
16. Improper Neutralization of Special Elements used in a Command ("Command Injection")
17. Improper Restriction of Operations Within the Bounds of a Memory Buffer
18. Use of Hard-coded Credentials
19. Server-Side Request Forgery (SSRF)
20. Missing Authentication for Critical Function
21. Concurrent Execution Using Shared Resource With Improper Synchronization ("Race Condition")
22. Improper Privilege Management
23. Improper Control of Generation of Code ("Code Injection")

Incorrect Authorization

25. Incorrect Default Permissions

Supply Chain Vulnerabilities

- Solar Winds
  - Organizational & process “bug”
  - Not detectable by tools

- Software as a Service (SaaS) & Continuous Integration/Continuous Deployment (CI/CD) move faster
  - Pushing potential harm out faster
  - Potentially erases security boundaries between “Dev” & “Ops”

- Without Software Bill Of Materials (SBOM), we are flying 90% blind
  - Even *with* SBOM, *someone* must own and fix or control the vulnerabilities

PC Firmware Supply Chain Vulnerabilities

- **CVE-2021-42059, CVSS score 7.5 - 8.2**

Common Vulnerabilities and Exposures, Common Vulnerability Scoring System

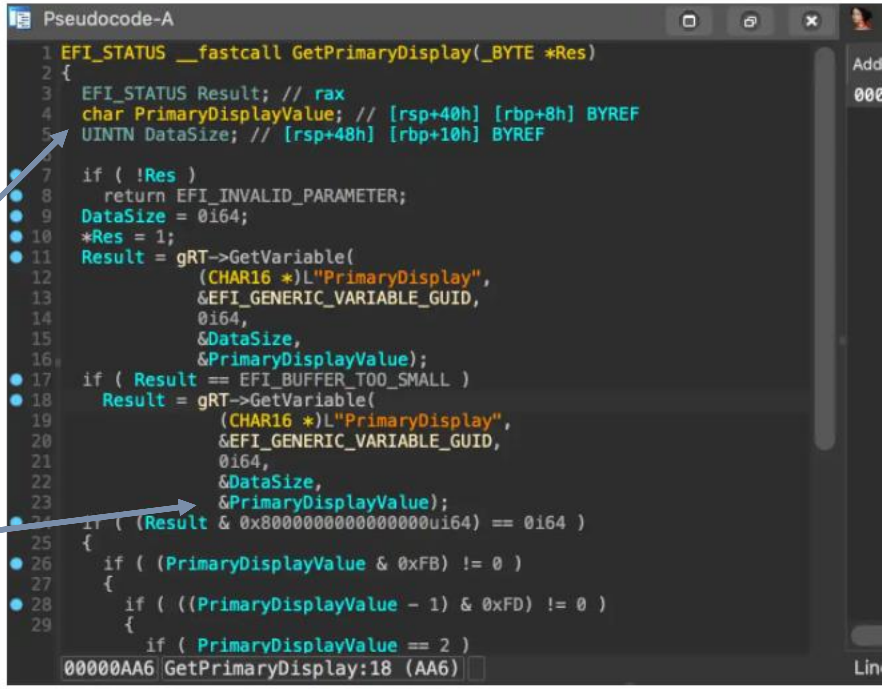
- **System Management Module in the UEFI (BIOS)**

Unified Extensible Firmware Interface
Basic Input Output System

- **Dozens of PC vendors affected**
- **Compromise not detectable by firmware integrity monitoring systems**

Allocate 1 byte

Read up to 64 bytes



```
EFI_STATUS __fastcall GetPrimaryDisplay(_BYTE *Res)
{
  EFI_STATUS Result; // rax
  char PrimaryDisplayValue; // [rsp+40h] [rbp+8h] BYREF
  UINTN DataSize; // [rsp+48h] [rbp+10h] BYREF

  if ( !Res )
    return EFI_INVALID_PARAMETER;
  DataSize = 0i64;
  *Res = 1;
  Result = gRT->GetVariable(
             (CHAR16 *)L"PrimaryDisplay",
             &EFI_GENERIC_VARIABLE_GUID,
             0i64,
             &DataSize,
             &PrimaryDisplayValue);
  if ( Result == EFI_BUFFER_TOO_SMALL )
    Result = gRT->GetVariable(
               (CHAR16 *)L"PrimaryDisplay",
               &EFI_GENERIC_VARIABLE_GUID,
               0i64,
               &DataSize,
               &PrimaryDisplayValue);
  if ( (Result & 0x8000000000000000ui64) == 0i64 )
  {
    if ( (PrimaryDisplayValue & 0xFB) != 0 )
    {
      if ( ((PrimaryDisplayValue - 1) & 0xFD) != 0 )
      {
        if ( PrimaryDisplayValue == 2 )
```

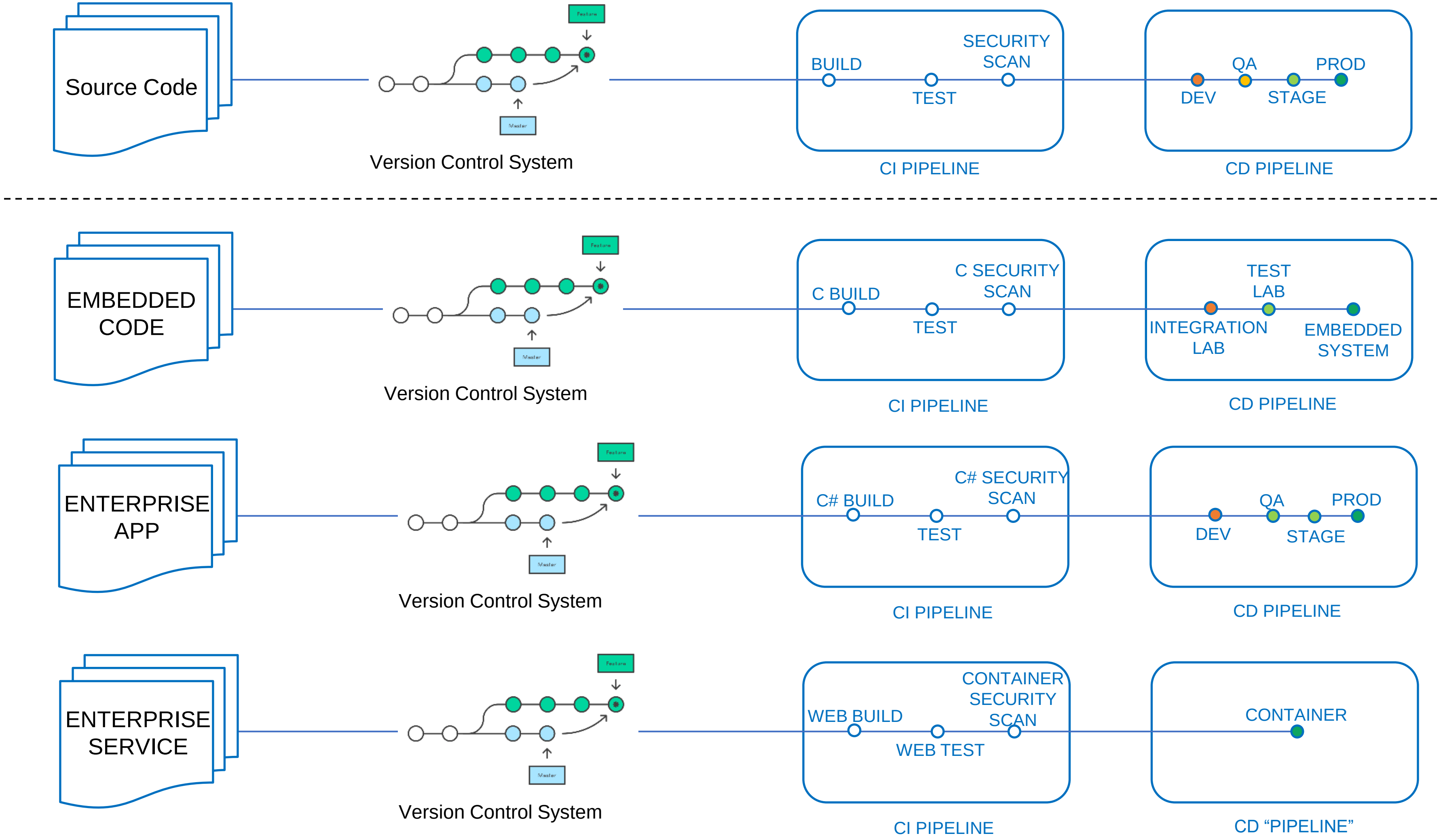
```
00000AA6 GetPrimaryDisplay:18 (AA6)
```

https://www.binarly.io/posts/An_In_Depth_Look_at_the_23_High_Impact_Vulnerabilities/index.html

“The” DevSecOps Pipeline

- **Idea**

- **Reality**



Briefer: Lynn Wallace (517 SWES/MXDPA)

Code With Honor

How to Fix

The Solution

Ensure that everyone who creates software knows secure coding principles - worldwide.

Help Is Available

- **Government**
  - Joint Federated Assurance Center (JFAC), https://jfac.dso.mil
  - Cyber Resiliency Office for Weapon Systems (CROWS)
  - National Institute of Standards and Technology (NIST) Software and Supply Chain Assurance (SSCA) Forum and Working Groups
  - DOD/NNSA Software Assurance Community of Practice
- **Academia**
  - Mr. John Keane ("The Software Angel of Death")
  - Nancy Mead (Software Engineering Institute, former)
  - Carol Woody (Software Engineering Institute)
  - David Wheeler (Linux Foundation)
- **Private Sector**
  - SAFECode, Secure Software Alliance, CISQ, OWASP, BSIMM

What You Should Do

- **Begin teaching secure design principles to your programmers ASAP**
  - ***Software Assurance* certification via Defense Acquisition University (DAU) and ISC2**
  - **Start thinking like an attacker and learn about hacking (practice carefully!)**
  - **Assume: The enemy is calling *your* function**
  - **Point them to CWE, OWASP, Known Exploited Vulnerabilities (KEVs)**
- **Enforce DevSecOps with security scans**
  - **Pick scanning tools that teach**
  - **Don't just count the weaknesses, *fix them***
- **Connect Cyber, Program Protection, Mission Defense to SW teams**
  - **Requirements, Oversight, Mentoring and Collaboration**
    - **Data Flow**
    - **Misuse/abuse analysis as informed by threats**
    - **Metrics or other evidence**
    - **Don't forget Supply Chain Risk Management (SCRM)**

Questions?

Software Assurance Is:

Definitions

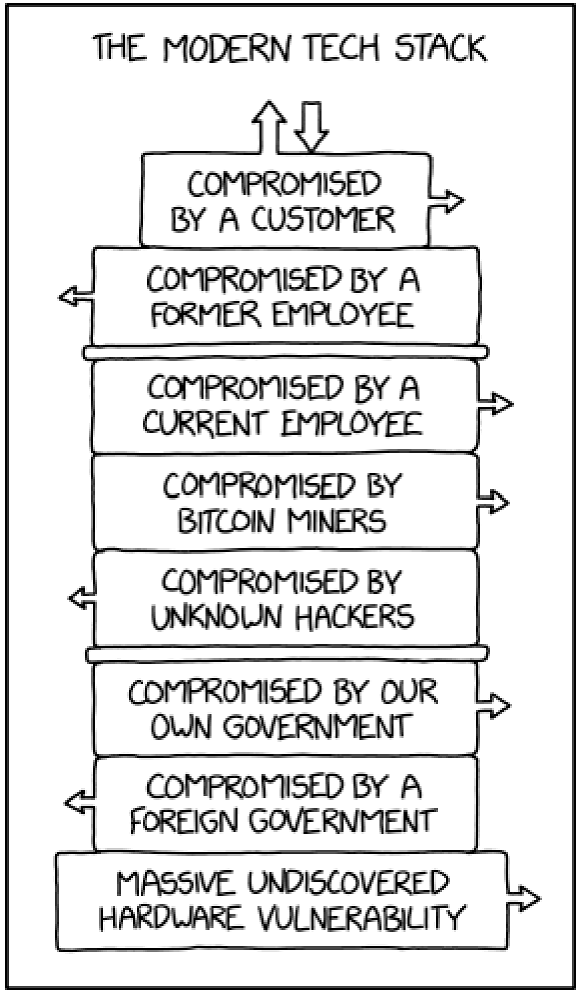
The level of confidence that software functions as intended and is free of vulnerabilities, either intentionally or unintentionally designed or inserted as part of the software throughout the lifecycle. ~DODI 5200.44

Cybersecurity is defined as:

Prevention of damage to, protection of, and restoration of computers, electronic communications systems, electronic communications services, wire communication, and electronic communication, including information contained therein, to ensure its availability, integrity, authentication, confidentiality, and nonrepudiation. ~DoDI 8500.01

Abbreviated as "SwA"

50-Year-Old Supply Chain


THE MODERN TECH STACK
COMPROMISED BY A CUSTOMER
COMPROMISED BY A FORMER EMPLOYEE
COMPROMISED BY A CURRENT EMPLOYEE
COMPROMISED BY BITCOIN MINERS
COMPROMISED BY UNKNOWN HACKERS
COMPROMISED BY OUR OWN GOVERNMENT
COMPROMISED BY A FOREIGN GOVERNMENT
MASSIVE UNDISCOVERED HARDWARE VULNERABILITY

https://xkcd.com/2166

Saltzer & Schroeder, 1974

Principle	Lesson	Examples
Economy of mechanism	KISS (Keep It Simple, Stupid)	Single sing-on, password vaults, resource efficiency
Fail-safe defaults	A fault in a “default deny” system is easily detected: “WHY DON’T I HAVE ACCESS ANYMORE?” A fault in a “default allow” system hides until exploited.	log4shell
Complete mediation	Every access to every object must be checked for authorization.	Cookie management, session management, caching of credentials
Open design	“Security by obscurity” does not work.	Kerckhoff’s principle, peer review, open source, crowd source
Separation of privilege/duties	Where feasible, a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key.	Multi-party tasks, secret sharing, split knowledge
Least privilege	Every program and every user of the system should operate using the least set of privileges necessary to complete the job.	Access control, need-to-know, run-time privileges
Least common mechanism	Minimize the amount of mechanism common to more than one user and depended on by all users.	Compartmentalization/isolation, allow-accept list
Psychological acceptability	It is essential that the human interface be designed for ease of use so that users routinely and automatically apply the protection mechanisms correctly.	Password complexity, passwordless authentication, screen layouts, Completely Automated Public Turing test to tell Computers and Humans apart (CAPTCHA)
Work factor	Compare the cost of circumventing the mechanism with the resources of a potential attacker. “How valuable is your information?” “To you?” “To an attacker?”	All security measures
Compromise recording	Provide diagnostics but beware of your reader!	Logging
Component reuse	Do not create your own encryption, authentication, etc.	Common controls, libraries
Resiliency	Resist compromise, quickly return to normal after attack.	Fail safe, fail secure, no single point of failure, failover
Defense in depth	Apply these principles everywhere.	Layered controls, geographical diversity, technical diversity, distributed systems