Secure Programming Lab

A.A. 2022/2023

Corso di Laurea in Ingegneria delle Telecomunicazioni

A. Introduction

Paolo Ottolino

Politecnico di Bari

DIPARTIMENTO DI
INGEGNERIA ELETTRICA
E DELL'INFORMAZIONE

Secure Programming Lab

Learning Objectives

1. **Getting:**
   - Cyber Security: **Protection needs**
   - Secure Programming: **key Practices**
   - Nowadays architecture: **Development methodologies**

   That should be involved every time enterprise application is developed.

2. **Building the foundation for implementing DevSecOps**

3. **… and also understand enterprise applications in order to better integrate with**
   - the operational environment
   - the most possible already developed components
   - The business environment

This **course** collects and merges **information** from **many sources**

Secure Programming Lab: Course Program

A. **Intro Secure Programming: «Who-What-Why-When-Where-How»**
B. **Building Security in: Buffer Overflow, UAF, Command Inection**
C. **Architecture and Processes: App Infrastructure, Three-Tiers, Cloud, Containers, Orchestration**
D. **SwA (Software Assurance): Vulnerabilities and Weaknesses (CVE, OWASP, CWE)**
E. **Security & Protection: Risks, Attacks. CIA -> AAA (AuthN, AuthZ, Accounting) -> IAM, SIEM, SOAR**
F. **Architecture and Processes 2: Ciclo di Vita del SW (SDLC), DevSecOps**
G. **Dynamic Security Test: VA, PT, DAST (cfr. VulnScanTools), WebApp Sec Scan Framework (Arachni, SCNR)**
H. **Free Security Tools: OWASP (ZAP, ESAPI, etc), NIST (SAMATE, SARD, SCSA, etc), SonarCube, Jenkins**
I. **Architecture and Processes 3: OWASP DSOMM, NIST SSDF**
J. **Operating Environment: Kali Linux on WSL**
K. **Python: Powerful Language for easy creation of hacking tools**
L. **SAST: Endogen, Exogen factors, SAST (cfr. SourceCodeAnalysisTools), SonarQube**
M. **Exercises: SecureFlag**

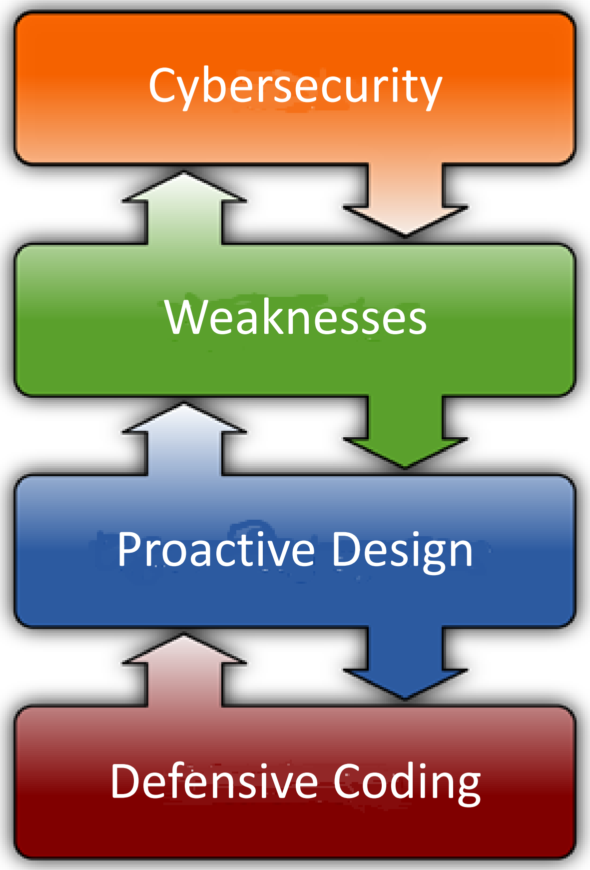
Secure Programs: Introduction

1. **Secure Programming**: Introduction
2. **Cyber Threats**: a perspective
3. **Weaknesses**: Tools (OWASP Top10 Cyber Kill Chain , Glossary (security elements of an attack)
4. **Secure Design**: Best Practices (NIST CSF, ZTA, DevSecOps)
5. **Code Vulnerabilities**: Buffer Overflow, Insecure Input

A.1 Secure Programming: Introduction

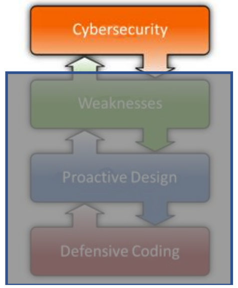
Secure Programming

1. **Secure Programming**: developing software in such a way to reduce the probability of damages from any usage
2. **Cybersecurity** (why): **reducing the risk** (ideally eliminating the possibility) that the **applications** could be **exploited** through **cyber-threats**
3. **Weaknesses** (what): removing defects in **architecture** and **software** that can be exploited to **attack** companies and its computer systems
4. **Proactive Design** (where): integrate the **architecture** so that applications can operate more safely
5. **Defensive Coding** (how): developing application in such a way that **guards against** the accidental introduction of **software vulnerabilities**
6. **Official Birthday** (when): November 22nd , 1988 (Morris Worm)



A.1b Secure Programming: Introduction

Cybersecurity (why): risk of cyber-threats



Quantitative Risk == **ARO** x **SLE**

probability (ARO) of loosing money (SLE) from incidents or attacks (Threats) by exploiting 1+ vulnerability.

Usually, the security risk is calculated on an annual basis

The overall Risk is the combination of all the single impacts.

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

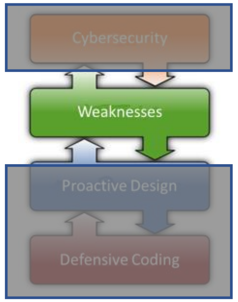
Qualitative Risk (e.g. OWASP Risk Methodology)

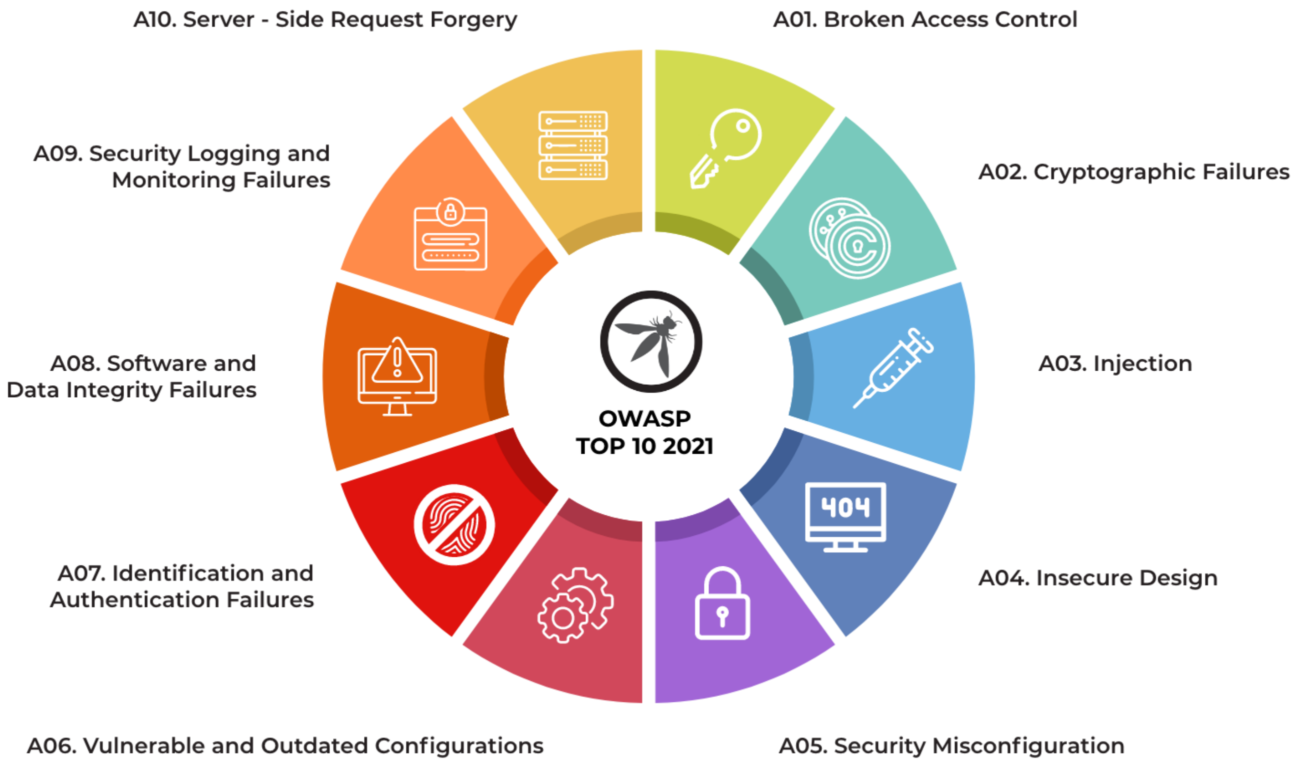
ARO: **Annual Rate of Occurrence** → Likelihood (probability), external factor: **threat**

SLE: **Single Loss Expectancy** → Impact (money), internal factor: **vulnerability**

A.1c Secure Programming: Introduction

Weaknesses (what): removing exploitable defects in software and architecture





(Open Web Application Security Project) OWASP Top 10
The 10 most important and frequent vulnerabilities identified 2017-2021

A vulnerability is a hole or a weakness in the application, which can be a design flaw or an implementation bug, that allows an attacker to cause harm to the stakeholders of an application.

Stakeholders include the application owner, application users, and other entities that rely on the application.

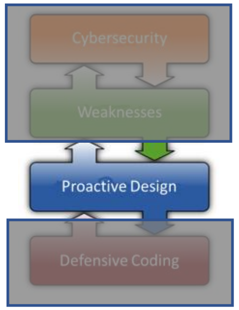
Examples:

- Lack of input validation on user input
- Lack of sufficient logging mechanism
- Fail-open error handling
- Not closing the database connection properly

For a great overview, check out the OWASP Top Ten Project.

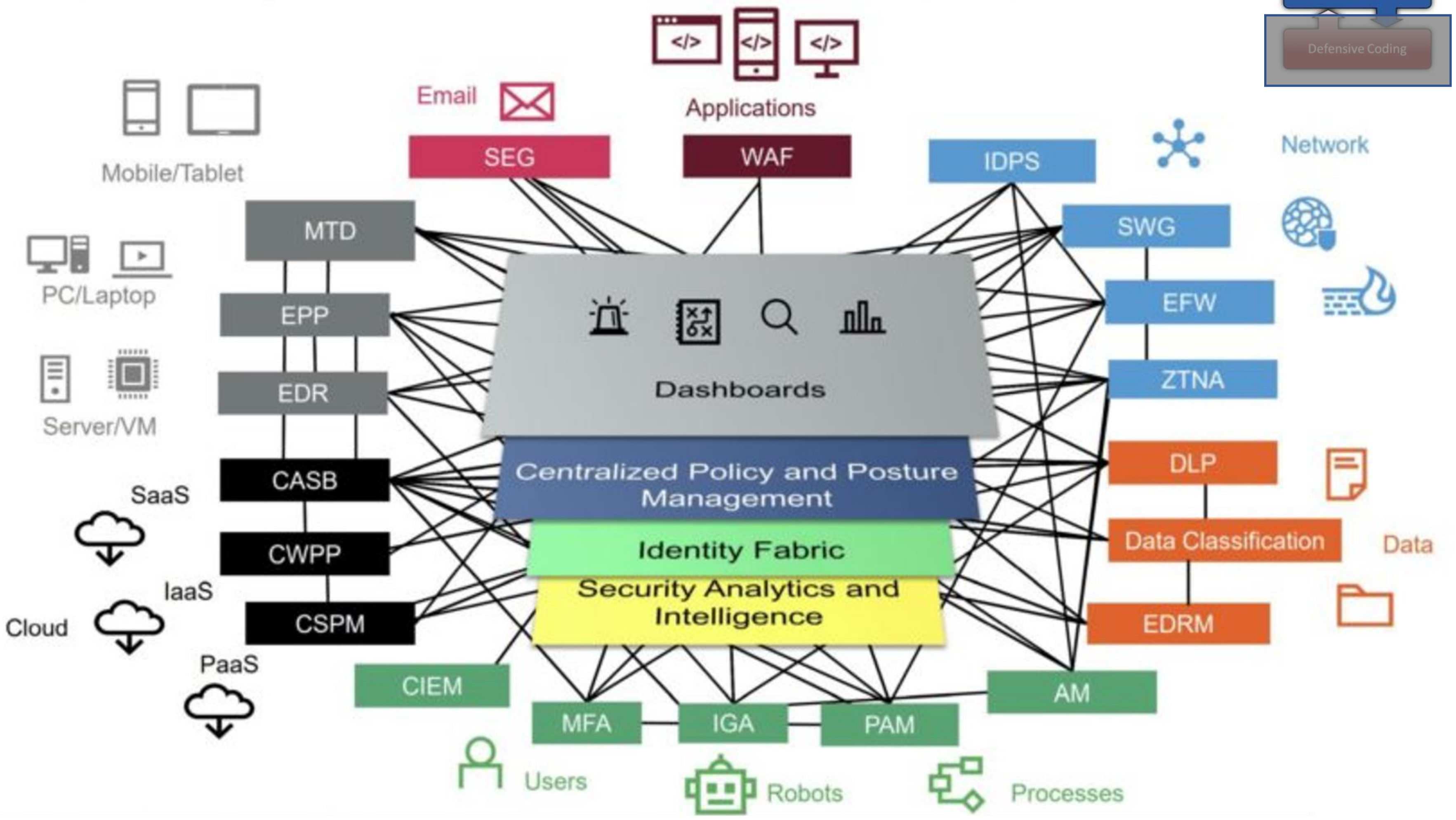
A.1d Secure Programming: Introduction

Proactive Design (where): safer architecture integration



Nowadays application software should guarantee interoperability, that is the ability to communicate and share information about cybersecurity.

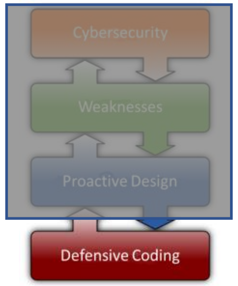
No more silos: every component is part of a bigger infrastructure, giving some service and obtaining some other back.



Gartner CSMA: Cyber Security Mesh Architecture

A.1e Secure Programming: Introduction

Defensive Coding (how): developing without security bugs



The first bug (Source: Naval Historical Center Online Library Photograph)

The causes of security breaches are varied, but many of them owe to a defect (or **bug**) or design flaw in a targeted computer system's software.

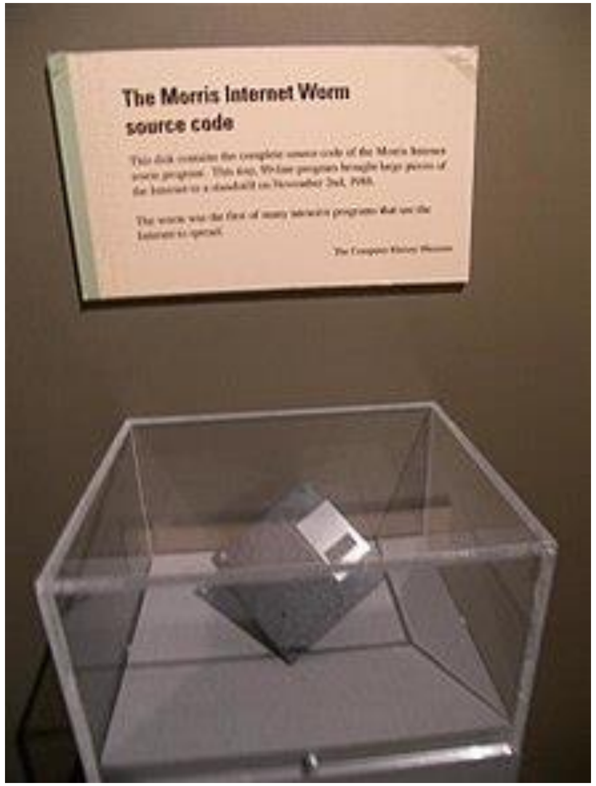
After finding a moth inside the Harvard Mark II computer on September 9th, 1947 at 3:45 p.m., Grace Murray Hopper logged the first computer bug in her log book.

She wrote the time and the sentence: “First actual case of bug being found”.

Nowadays, the term “bug” in computer science is not taken literally, of course. We use it to talk about a flaw or failure in a computer program that causes it to produce an unexpected result or crash.

A.1f Secure Programming: Introduction

Official Birthday (when): November 22°, 1988 (Morris Worm)



Floppy disk containing the source code for the Morris Worm, at the Computer History Museum

The **Morris worm** or **Internet worm of November 2, 1988**, is one of the oldest computer worms distributed via the Internet, and the first to gain significant mainstream media attention.

It resulted in the first felony conviction in the US under the 1986 Computer Fraud and Abuse Act.

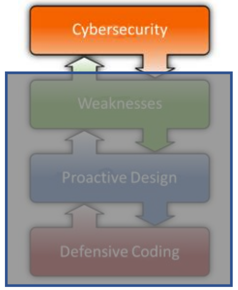
It was written by a graduate student at Cornell University, Robert Tappan Morris, and launched on 8:30 pm November 2, 1988, from the Massachusetts Institute of Technology network.

The worm exploited several vulnerabilities of targeted systems, including:

- A hole in the debug mode of the Unix sendmail program
- A buffer overflow or overrun hole in the finger network service
- The transitive trust enabled by people setting up network logins with no password requirements via remote execution (rexec) with Remote Shell (rsh), termed rexec/rsh

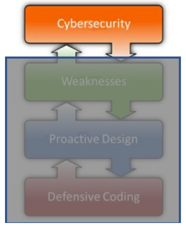
A.2 Cyber Threats: a perspective

Reduce Losses, Know Occurrences



A.1b Secure Programming: Introduction

Cybersecurity (why): risk of cyber-threats



Quantitative Risk == **ARO** x **SLE**

probability (ARO) of loosing money (SLE) from incidents or attacks (Threats) by exploiting 1+ vulnerability.

Usually, the security risk is calculated on an annual basis

The overall Risk is the combination of all the single impacts.

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

Qualitative Risk (e.g. OWASP Risk Methodology)

ARO: **Annual Rate of Occurrence** → Likelihood (probability), external factor: **threat**
SLE: **Single Loss Expectancy** → Impact (money), internal factor: **vulnerability**

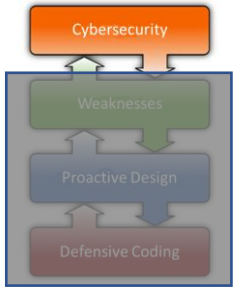
SLE could be **reduced**, working on **vulnerabilities** (internal factors)

ARO could be only **known** since it depends basely on **threats** (external factors)

➔ Sun Tzu Ping Fa

A.2 Cyber Threats: a perspective

Reduce Losses, Know Occurrences



Sun Tzu Ping Fa

“If you know the enemy (**ARO**) and know yourself (**SLE**), you need **not fear** the result of a hundred battles.

If you know yourself (**SLE**) but not the enemy (**~~ARO~~**), for every victory gained you will also **suffer a defeat**.

If you know neither the enemy (**~~ARO~~**) nor yourself (**~~SLE~~**), you will **succumb in every battle**.”

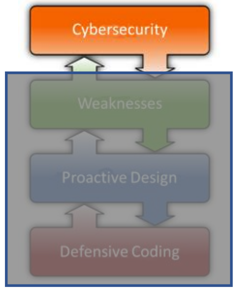
(from ch. III “Attack by Stratagems”, #18)

SLE ➔ Vulnerabilities: combination of Business and the 3 remaining layers (“Weaknesses”, “Proactive Design” and “Defensive Coding”.
ARO ➔ Threats: external factors

Let’s **have a look** at ARO ➔ (Cyber) **Threats**

A.2a Cyber Threats: a perspective

FBI Attacker Profiles



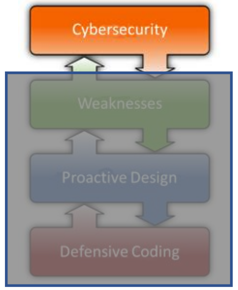
Cyber Threat Actors

Unstructured		**Insider**	Money
Structured		**Crime**	Money
		**Espionage**	Information
		**Hactivism**	Socio-Politics
National		**Warfare**	War
		**Terrorism**	War

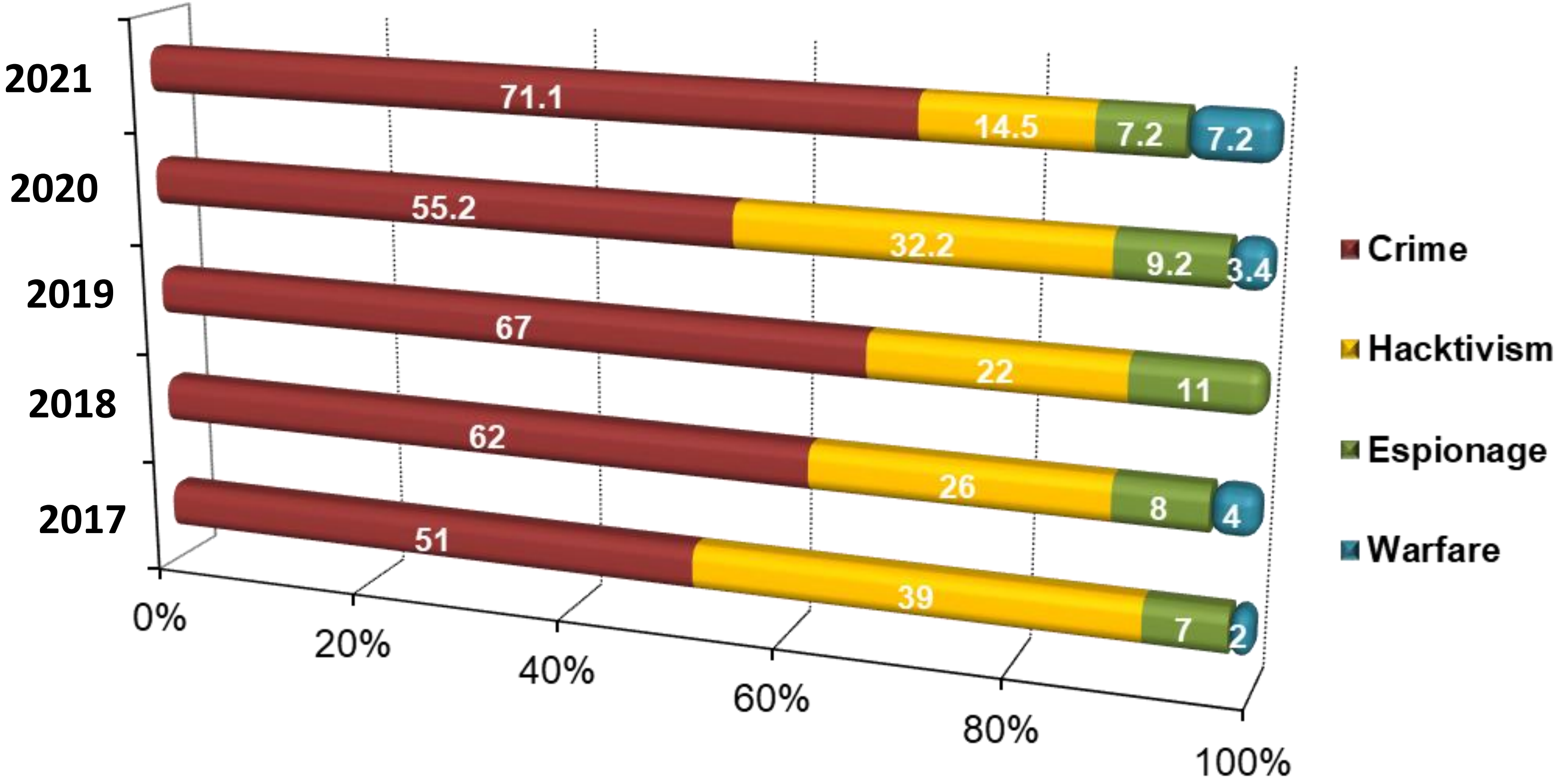
See «An introduction to the cyber threat environment»
https://cyber.gc.ca/en/guidance/introduction-cyber-threat-environment

A.2b Cyber Threats: a perspective

Cyber Threats: Historical Trends

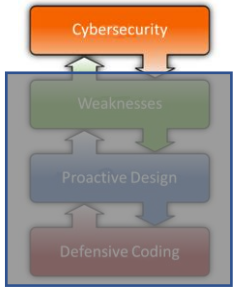


Percentages



A.2c Cyber Threats: a perspective

Exploting, Profiteering, Wasting



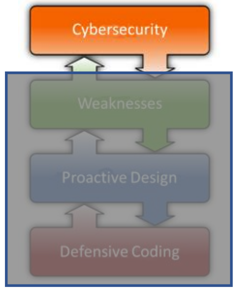
Exploiting, Profiteering, Wasting

- **Exploiting (Intruding): access system in order to:**
  - Control the performed actions
  - Harvest Information

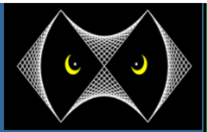
- **Profiteering: access to system, in order to take advantage from:**
  - elaboration
  - network capacities (to 3° parties)

- **Wasting (Damaging): make the system not accessible from anyone**

A.2d Cyber Threats: a perspective

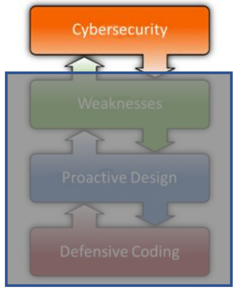


Adversary-Risk mapping (exemplification)

	 **Crime**	 **Hacktivism**	 **Warfare**	 **Espionage**
 **Intruding**	Steal Money Read User-Info	Steal Info	Steal Info	Steal Info
 **Profiteering**	Spam DDoS (3° party)			
 **Damaging**	DDoS (competitors)	Defacement	Break System	
	71%	15%	7%	7%

A.2e Cyber Threats: a perspective

Cyber Attack to Clients 1/3

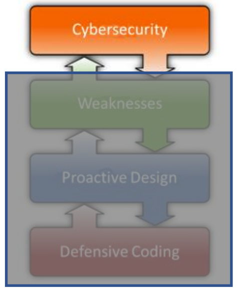


Motives

Motives	
BotNet	Network of computers compromised by malware and controlled remotely for illegal purposes. You join a botnet unknowingly when your computer is not properly protected and updated. Botnets pose an insidious threat as an infection can remain undetected and silent for a long time to be exploited later to produce massive damage to third-party systems
Ramsonware	Restricting access to the resources hosted by an infected device, demanding a ransom to be paid to remove it
Tailored	Set of stealthy and continuous cyber hacking processes, specially orchestrated to target a specific entity, damaging only systems with particular requirements

A.2f Cyber Threats: a perspective

Cyber Attack to Clients 1/3

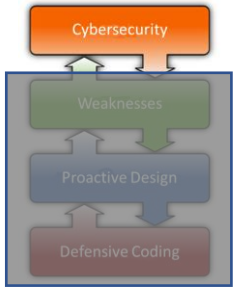


Means

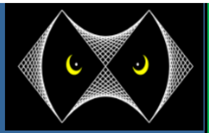
Infection	process of subjecting a system, perpetrated in one of the following ways:
Phishing	opening infected emails or documents attached to them
Malware	hidden in programs downloaded by users (e.g. cracks), aimed at disturbing the normal functioning of a system
Known Vulnerabilities	exploit specific vulnerabilities of out-of-date systems and applications

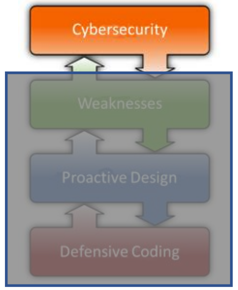
A.2g Cyber Threats: a perspective

Cyber Attack to Clients 3/3



Adversary-Attack mapping (exemplification for clients)

	 Crime	 Hacktivism	 Warfare	 Espionage
 **Intruding**	Ramsonware Tailored		Tailored	Tailored
 **Profiteering**	BotNet			
	86%		5%	9%



A.2h Cyber Threats: a perspective

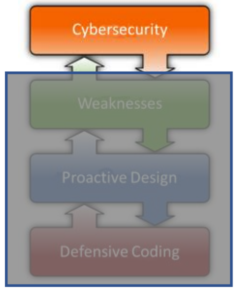
Historical Background: Operazione Mariposa (2009)

BotNet/Crime: 13 milions systems in 190+ countries

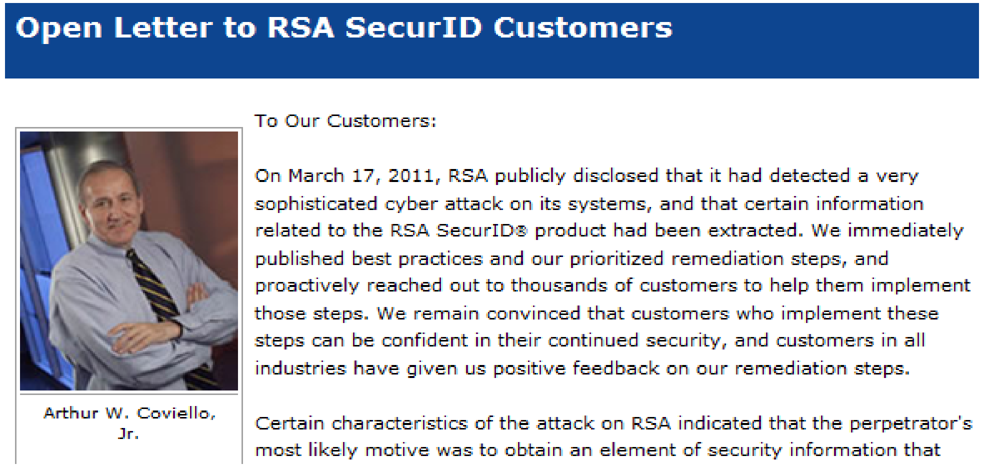
Working	Diffusion: developed using the Butterfly kit, a software package sold online for between €500-1500, with which 10,000 unique software packages have been created and around 700 BotNets built (in addition to Mariposa)
Scope	Used mainly for: • DDoS (BlackEnergy) • Hijacking (DNS poisoning) • Banking

A.2i Cyber Threats: a perspective

Historical Background: RSA SecurID Breach (2011)



Tailored/Espionage (Crime)



Open Letter to RSA SecurID Customers

Arthur W. Coviello, Jr.

To Our Customers:

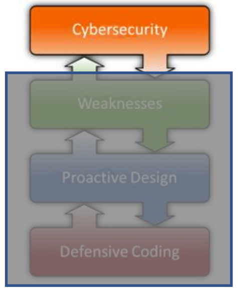
On March 17, 2011, RSA publicly disclosed that it had detected a very sophisticated cyber attack on its systems, and that certain information related to the RSA SecurID® product had been extracted. We immediately published best practices and our prioritized remediation steps, and proactively reached out to thousands of customers to help them implement those steps. We remain convinced that customers who implement these steps can be confident in their continued security, and customers in all industries have given us positive feedback on our remediation steps.

Certain characteristics of the attack on RSA indicated that the perpetrator's most likely motive was to obtain an element of security information that

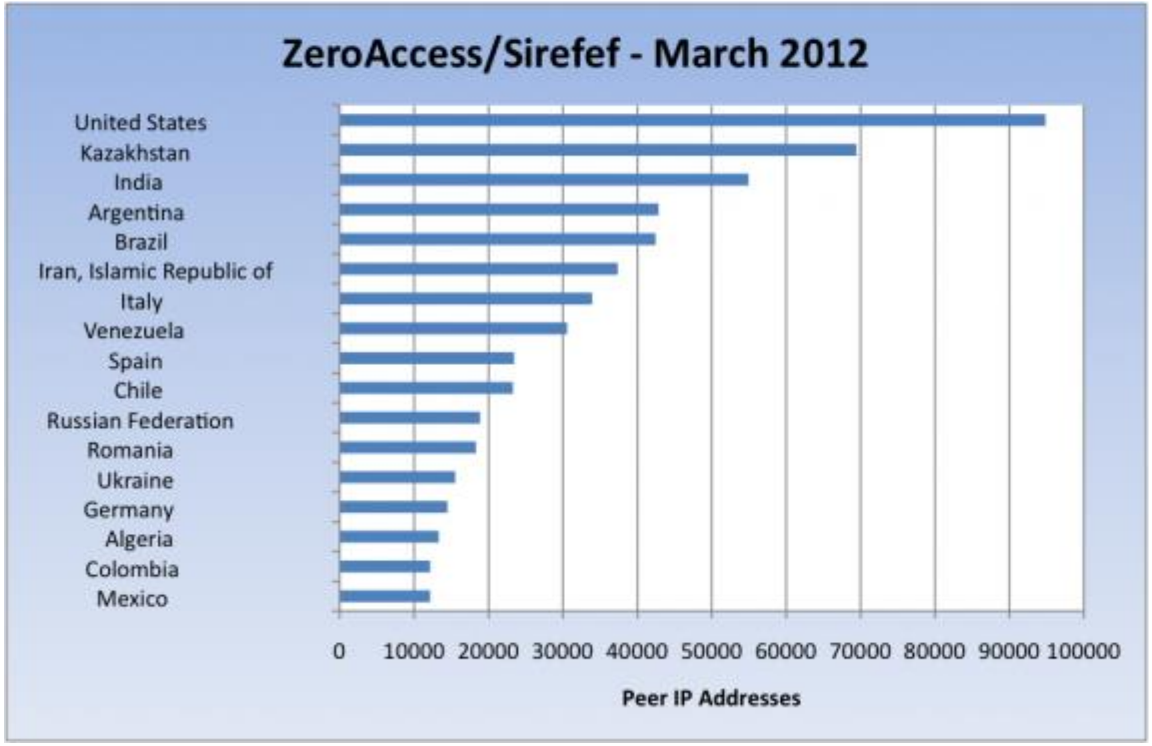
Working	The attack took place in several stages: 1. Collection of company information 2. Creation of a Phishing email, titled “2011 Recruitment Plan” and containing an xls attachment “2022 Recruitment plan 2011.xls”, containing a “zero-day” exploit 3. Determination of 2 (small) groups of RSA employees, potential "good" victims 4. Sending the first phih email to the first group 5. Sending the second phishing email to the second group 6. Malicious Code Execution Some user has installed the backdoor (Poison Ivy Trojan). 7. Privilege escalation 8. Access to servers containing SecurID key management information 9. Sending information to external servers and deleting information from RSA servers
Scope	Exfiltrate data from RSA to invalidate the OTP authentication mechanisms provided by devices generally used for Web Banking

A.2j Cyber Threats: a perspective

Historical Background: Zero Access (2013)



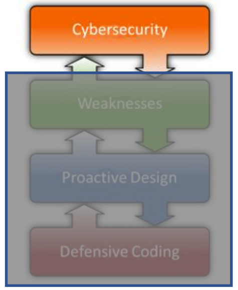
BotNet/Crime: 2 milions of systems – current most «popular» BotNet



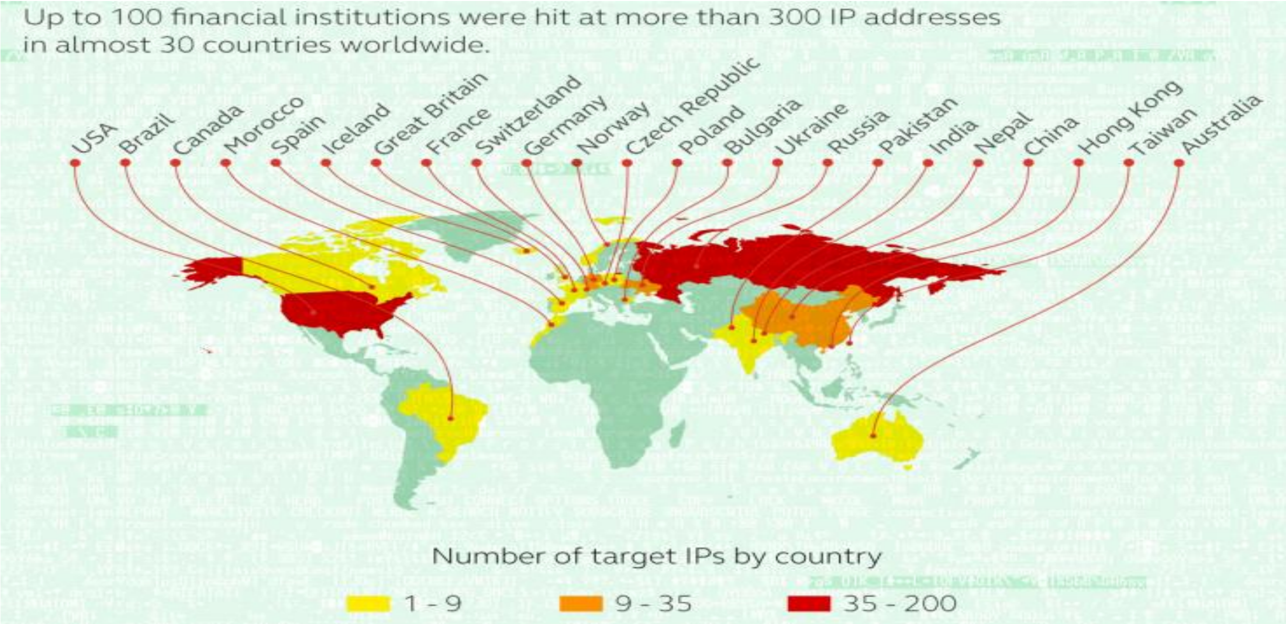
Working	Robustness: architecture based on Peer-2-Peer logic (resilience to destruction)
Scope	Earn from advertisements, through: • search results hijacking (Google, Bing, Yahoo) • redirection to unsolicited sites

A.2k Cyber Threats: a perspective

Historical Background: Carbanak (2015)



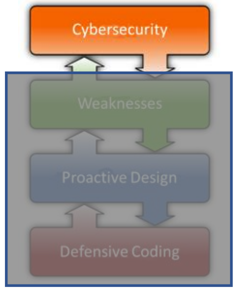
Tailored/Espionage-Crime: $ 1 Billion booty



Working	The attack took place in several stages: 1. Sending malware by email 2. Gain control of some locations 3. Study of the behavior of employees who operate money transfers 4. Study of the work of IT personnel, to access the central DB 5. play money transfer: adding a "0" to the balance of a low active customer and transferring the created funds 6. theft at ATMs with local solicitors
Scope	Collecting physical money: • Exempt from ATMs (Windows XP hosts) • Collected via the SWIFT network

A.2l Cyber Threats: a perspective

Historical Background: Hack Back (USA) - Active Cyber Defense Certainty Act of 2019



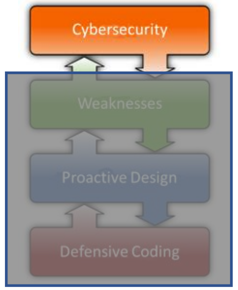
Hack Back - Conferimento dei poteri di Contrasto (Hack-Back) all''Intelligence italiana

Proposed Amendment in 2017 by Tom Graves
ACDC act «Highway to Hell»: https://www.congress.gov/bill/116th-congress/house-bill/3270

Working	To receive this type of waiver, companies must notify the FBI (National Cyber Investigation Joint Task Force): 1. Details about the counterattack tools in possession 2. How evidence of the initial cyber intrusion is kept 3. Methodologies and mechanisms with which it is intended to avoid damaging the systems of unarmed third parties
Finalità	Protect companies from legal prosecution should they proceed to fight back against the cyber attacker

A.2l Cyber Threats: a perspective

Historical Background: Hack Back (Italy) - Art. 37 del DL 11/2022 (“Aiuti bis”)



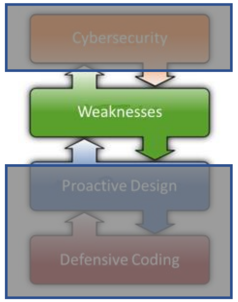
Hack Back - Conferimento dei poteri di Contrasto (Hack-Back) all’’Intelligence italiana

[**Art. 37 Disposizioni in materia di intelligence in ambito cibernetico**]

Funzionamento	Il Presidente del Consiglio dei ministri, acquisito il parere del Comitato interministeriale per la sicurezza della Repubblica e sentito il Comitato parlamentare per la sicurezza della Repubblica, emana disposizioni per l'adozione di misure di intelligence di contrasto in ambito cibernetico: 1. in situazioni di crisi o di emergenza 2. a fronte di minacce che coinvolgono aspetti di sicurezza nazionale 3. e non siano fronteggiabili solo con azioni di resilienza, 4. anche in attuazione di obblighi assunti a livello internazionale Tali misure sono attuate da AISI ed AISE
Finalità	Proteggere gli interessi e la sicurezza nazionali, autorizzando misure di contrasto in ambito cibernetico, scelte secondo criteri di necessità e proporzionalità al rischio calcolato

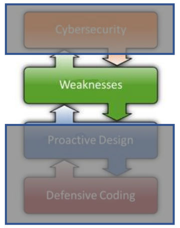
A.3 Weaknesses: Tools

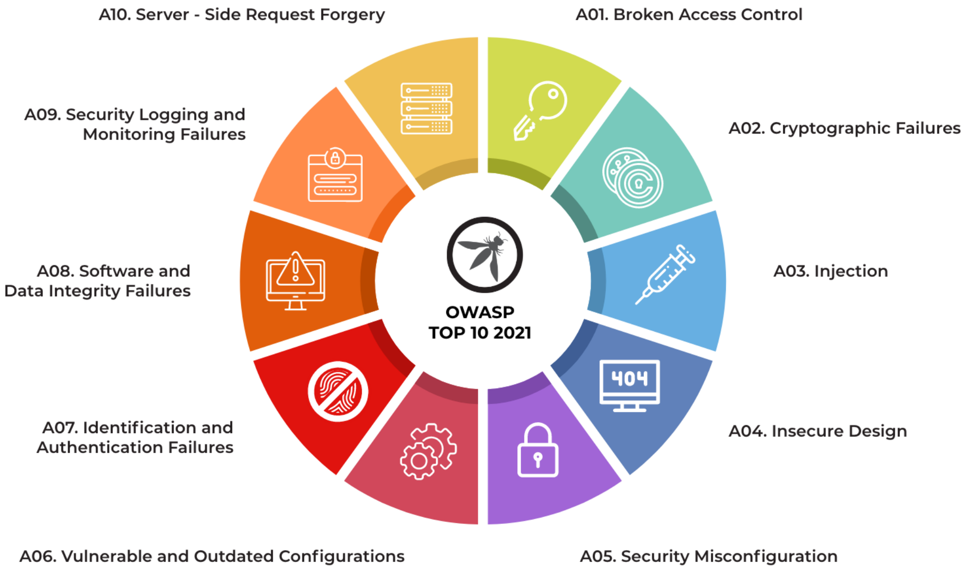
Introduction



A.1c Secure Programming: Introduction

Weaknesses (what): removing exploitable defects in software and architecture





(Open Web Application Security Project) OWASP Top 10
The 10 most important and frequent vulnerabilities identified 2017-2021

A vulnerability is a hole or a weakness in the application, which can be a design flaw or an implementation bug, that allows an attacker to cause harm to the stakeholders of an application.

Stakeholders include the application owner, application users, and other entities that rely on the application.

Examples:

- Lack of input validation on user input
- Lack of sufficient logging mechanism
- Fail-open error handling
- Not closing the database connection properly

For a great overview, check out the OWASP Top Ten Project.

According to Robert P. Cook, is hard to develop programs without bugs.

Some useful tools for avoiding inserting the most trivial ones, at least:

OWASP Top10: practical for Web App

CWE: taxonomy for more theoretycal purposes

CVE: common vulnerabilities in adopted platforms (and libraryies)

A.3a Weaknesses: Tools

Introduction

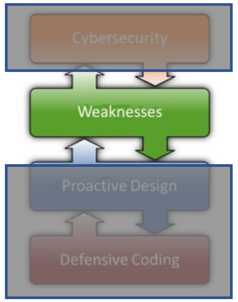
1. **OWASP Top10**: **de facto industry WebAppSec standard** (bare-minimum/starting-point for coding and testing). First one developed in 2003

2. **CWE**: **de facto weakness types standard for SW & HW** (taxonomy for classifying and defining weaknesses, in order to differentiate them). Established in 2006

3. **CVE**: **de facto vulnerability enumeration about COTS** (common vulnerability classification, in order to chose patched products). Presented in 1999

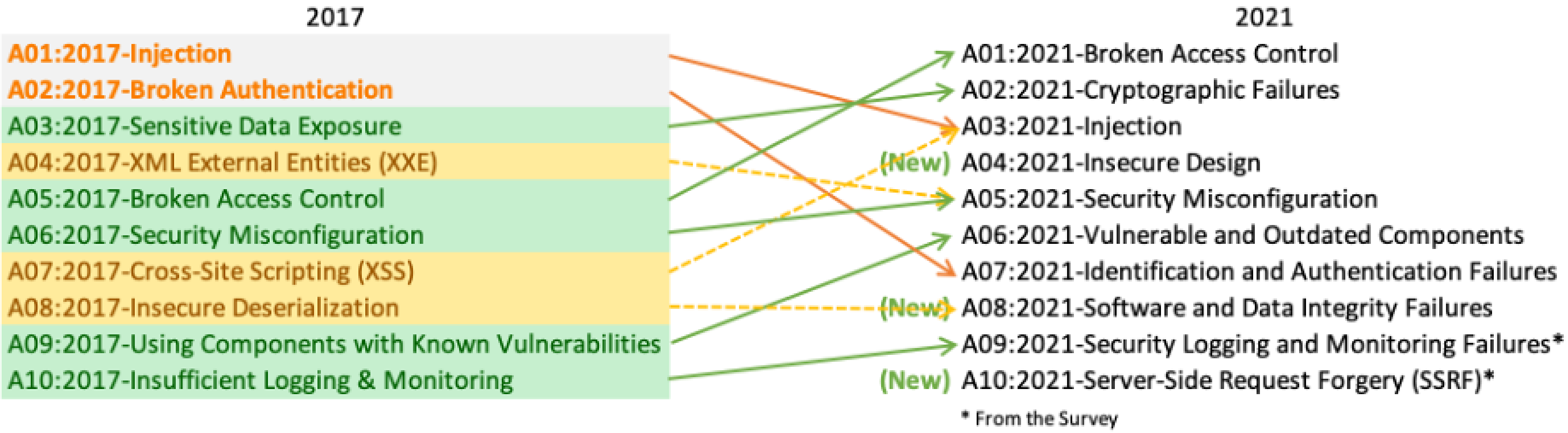
A.3.b Weaknesses: Tools

OWASP Top10



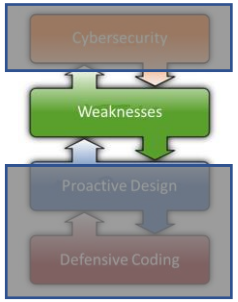
List of main 10 categories of vulnerabilities in Web Applications

- **Updated**: every 3-4 years
- **Web 2.0**: First published in 2003 (then 2004, 2007, 2010, 2013, 2017, 2021. see history)
- **Data Driven**: based on statistics about vulnerability assessment submission



A.3.b Weaknesses: Tools

OWASP Top10: Comparison of 2003, 2004, 2007, 2010 and 2013 Releases



OWASP Top Ten Entries (Unordered)	Releases				
	2003	2004	2007	2010	2013
Unvalidated Input	A1	A1[9]	×	×	×
Buffer Overflows	A5	A5	×	×	×
Denial of Service	×	A9[2]	×	×	×
Injection	A6	A6[3]	A2	A1[10]	A1
Cross Site Scripting (XSS)	A4	A4	A1	A2	A3
Broken Authentication and Session Management	A3	A3	A7	A3	A2
Insecure Direct Object Reference	×	A2	A4[11]	A4	A4
Cross Site Request Forgery (CSRF)	×	×	A5	A5	A8
Security Misconfiguration	A10	A10[3][5]	×	A6	A5
Missing Functional Level Access Control	A2	A2[1]	A10[13]	A8	A7[16]
Unvalidated Redirects and Forwards	×	×	×	A10	A10
Information Leakage and Improper Error Handling	A7	A7[14][4]	A6	A6[8]	×
Malicious File Execution	×	×	A3	A6[8]	×
Sensitive Data Exposure	A8	A8[6][5]	A8	A7	A6[17]
Insecure Communications	×	A10	A9[7]	A9	×
Remote Administration Flaws	A9	×	×	×	×
Using Known Vulnerable Components	×	×	×	×	A9 [18][19]

- X — removed
- [] — renamed
- (green) — ok

[1] Renamed “Broken Access Control” from T10 2003

[2] Split “Broken Access Control” from T10 2003

[3] Renamed “Command Injection Flaws” from T10 2003

[4] Renamed “Error Handling Problems” from T10 2003

[5] Renamed “Insecure Use of Cryptography” from T10 2003

[6] Renamed “Web and Application Server ” from T10 2003

[7] Split “Insecure Configuration Management” from T10 2004

[8] Reconsidered during T10 2010 Release Candidate (RC)

[9] Renamed “Unvalidated Parameters” from T10 2003

[10] Renamed “Injection Flaws” from T10 2007

[11] Split “Broken Access Control” from T10 2004

[12] Renamed “Insecure Configuration Management” from T10 2004

[13] Split “Broken Access Control” from T10 2004

[14] Renamed “Improper Error Handling” from T10 2004

[15] Renamed “Insecure Storage” from T10 2004

[16] Renamed “Failure to Restrict URL Access” from T10 2010

[17] Renamed “Insecure Cryptographic Storage” from T10 2010

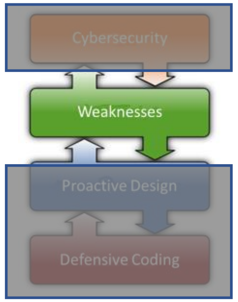
[18] Split “Insecure Cryptographic Storage” from T10 2010

[19] Split “Security Misconfiguration” from T10 2010

A.3c Weaknesses: Tools

OWASP Top10:2021

List of 10 main categories of vulnerabilities in Web Applications



A01:2021-Broken Access Control

A02:2021-Cryptographic Failures

A03:2021-Injection

A04:2021-Insecure Design

A05:2021-Security Misconfiguration

A06:2021-Vulnerable and Outdated Components

A07:2021-Identification and Authentication Failures

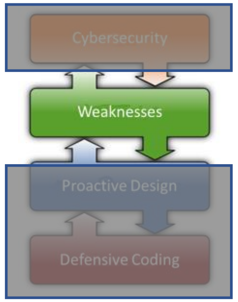
A08:2021-Software and Data Integrity Failures

A09:2021-Security Logging and Monitoring Failures

A10:2021-Server Side Request Forgery

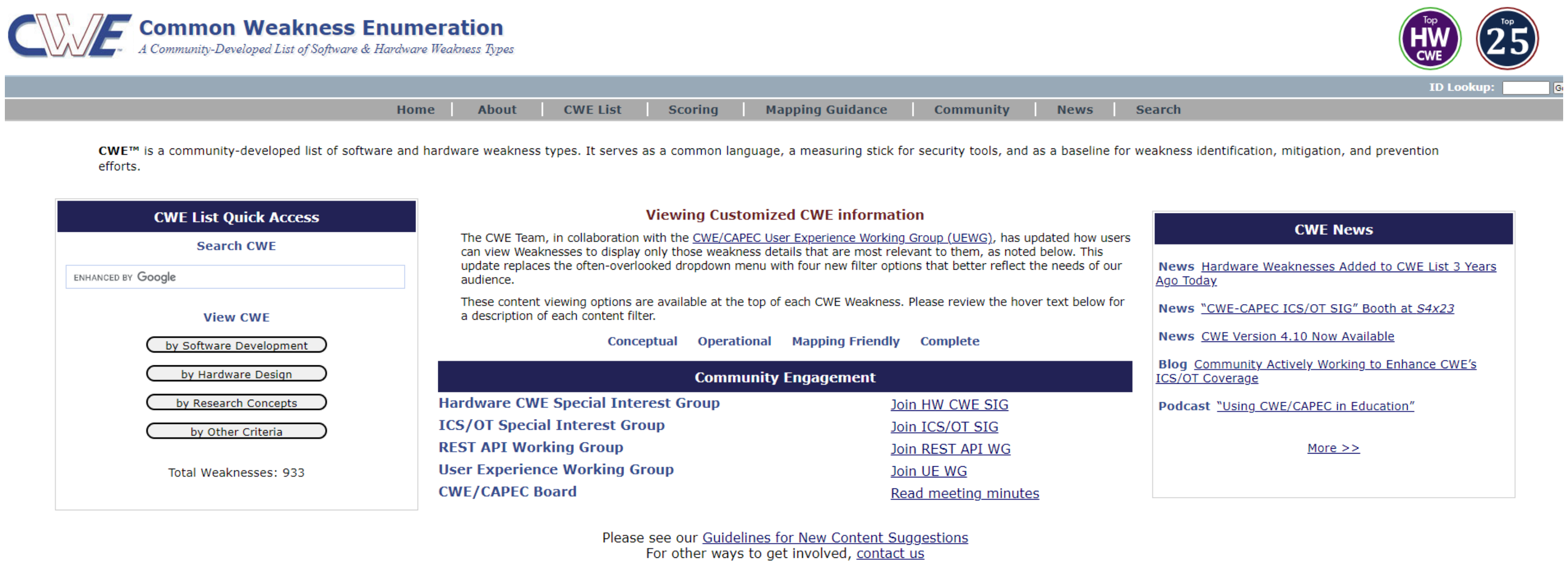
A.3d Weaknesses: Tools

MITRE: CWE



https://cwe.mitre.org

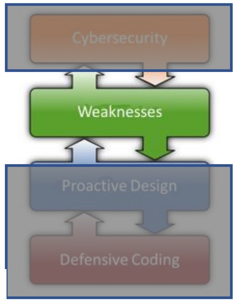
MITRE began working on the issue of categorizing software weaknesses as early 1999 when it launched the Common Vulnerabilities and Exposures (CVE®) List. As part of the development of CVE, MITRE's CVE Team developed a preliminary classification and categorization of vulnerabilities, attacks, faults, and other concepts to help define common software weaknesses.



However, while sufficient for CVE, those groupings are too rough to be used to identify and categorize the functionality offered within the offerings of the code security assessment industry. To support that type of usage, additional fidelity and succinctness are needed as are additional details and description for each of the different nodes and groupings such as the effects, behaviors, and implementation details, etc.

A.3d Weaknesses: Tools

MITRE: CWE Top 25 1/2



2022 CWE Top 25 Most Dangerous Software Weaknesses

Introduction

Welcome to the 2022 Common Weakness Enumeration (CWE™) Top 25 Most Dangerous Software Weaknesses list (CWE™ Top 25). This list demonstrates the currently most common and impactful software weaknesses. Often easy to find and exploit, these can lead to exploitable vulnerabilities that allow adversaries to completely take over a system, steal data, or prevent applications from working.

Many professionals who deal with software will find the CWE Top 25 a practical and convenient resource to help mitigate risk. This may include software architects, designers, developers, testers, users, project managers, security researchers, educators, and contributors to standards developing organizations (SDOs).

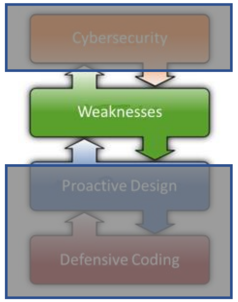
To create the list, the CWE Team leveraged Common Vulnerabilities and Exposures (CVE®) data found within the National Institute of Standards and Technology (NIST) National Vulnerability Database (NVD) and the Common Vulnerability Scoring System (CVSS) scores associated with each CVE Record, including a focus on CVE Records from the Cybersecurity and Infrastructure Security Agency (CISA) Known Exploited Vulnerabilities (KEV) Catalog. A formula was applied to the data to score each weakness based on prevalence and severity.

The dataset analyzed to calculate the 2022 Top 25 contained a total of 37,899 CVE Records from the previous two calendar years.

Table of Contents

A.3d Weaknesses: Tools

MITRE: CWE Top 25 2/2

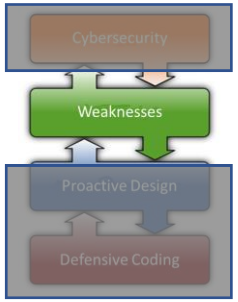


Rank	ID	Name	Score	KEV Count (CVEs)	Rank Change vs. 2021
1	CWE-787	Out-of-bounds Write	64.20	62	0
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.97	2	0
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	22.11	7	+3 ▲
4	CWE-20	Improper Input Validation	20.63	20	0
5	CWE-125	Out-of-bounds Read	17.67	1	-2 ▼
6	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	17.53	32	-1 ▼
7	CWE-416	Use After Free	15.50	28	0
8	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.08	19	0
9	CWE-352	Cross-Site Request Forgery (CSRF)	11.53	1	0
10	CWE-434	Unrestricted Upload of File with Dangerous Type	9.56	6	0
11	CWE-476	NULL Pointer Dereference	7.15	0	+4 ▲
12	CWE-502	Deserialization of Untrusted Data	6.68	7	+1 ▲
13	CWE-190	Integer Overflow or Wraparound	6.53	2	-1 ▼
14	CWE-287	Improper Authentication	6.35	4	0
15	CWE-798	Use of Hard-coded Credentials	5.66	0	+1 ▲
16	CWE-862	Missing Authorization	5.53	1	+2 ▲
17	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	5.42	5	+8 ▲
18	CWE-306	Missing Authentication for Critical Function	5.15	6	-7 ▼
19	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	4.85	6	-2 ▼
20	CWE-276	Incorrect Default Permissions	4.84	0	-1 ▼
21	CWE-918	Server-Side Request Forgery (SSRF)	4.27	8	+3 ▲
22	CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	3.57	6	+11 ▲
23	CWE-400	Uncontrolled Resource Consumption	3.56	2	+4 ▲
24	CWE-611	Improper Restriction of XML External Entity Reference	3.38	0	-1 ▼
25	CWE-94	Improper Control of Generation of Code ('Code Injection')	3.32	4	+3 ▲

The list of the weaknesses in the 2022 CWE Top 25, including the overall score of each. The KEV Count (CVEs) shows the number of CVE-2020/CVE-2021 Records from the CISA KEV list that were mapped to the given weakness.

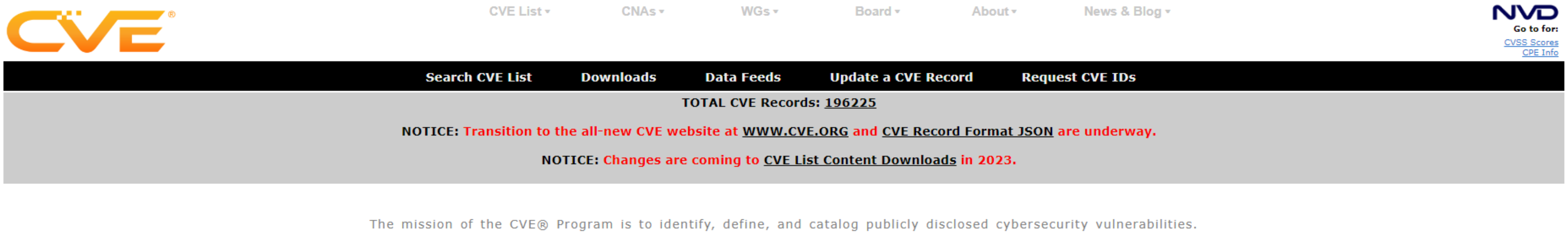
A.3d Weaknesses: Tools

MITRE: CVE



https://cve.mitre.org

The original concept for what would become the CVE List was presented by the co-creators of CVE, The MITRE Corporation's David E. Mann and Steven M. Christey, as a white paper entitled, ***Towards a Common Enumeration of Vulnerabilities (PDF, 0.3MB)***, at the *2nd Workshop on Research with Security Vulnerability Databases* on January 21-22, 1999 at Purdue University in West Lafayette, Indiana, USA.
From that original concept, a working group was formed (which would later become the initial 19-member CVE Editorial Board), and the original **321 CVE Records** were created. The CVE List was officially launched for the public in September 1999.

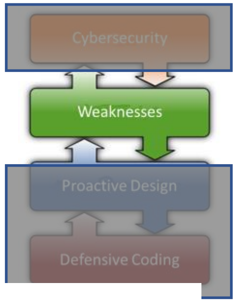


Nowadays (24 years later) there are about **200.000 CVE Records**

A.3d Weaknesses: Tools

MITRE: CVE Search

https://cve.mitre.org/cve/search_cve_list.html



Search Results

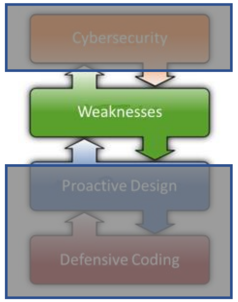
There are **22** CVE Records that match your search.

Name	Description
CVE-2022-33915	Versions of the Amazon AWS Apache Log4j hotpatch package before log4j-cve-2021-44228-hotpatch-1.3.5 are affected by a race condition that could lead to a local privilege escalation. This Hotpatch package is not a replacement for updating to a log4j version that mitigates CVE-2021-44228 or CVE-2021-45046; it provides a temporary mitigation to CVE-2021-44228 by hotpatching the local Java virtual machines. To do so, it iterates through all running Java processes, performs several checks, and executes the Java virtual machine with the same permissions and capabilities as the running process to load the hotpatch. A local user could cause the hotpatch script to execute a binary with elevated privileges by running a custom java process that performs exec() of an SUID binary after the hotpatch has observed the process path and before it has observed its effective user ID.
CVE-2022-29615	SAP NetWeaver Developer Studio (NWDS) - version 7.50, is based on Eclipse, which contains the logging framework log4j in version 1.x. The application's confidentiality and integrity could have a low impact due to the vulnerabilities associated with version 1.x.
CVE-2022-24818	GeoTools is an open source Java library that provides tools for geospatial data. The GeoTools library has a number of data sources that can perform unchecked JNDI lookups, which in turn can be used to perform class deserialization and result in arbitrary code execution. Similar to the Log4J case, the vulnerability can be triggered if the JNDI names are user-provided, but requires admin-level login to be triggered. The lookups are now restricted in GeoTools 26.4, GeoTools 25.6, and GeoTools 24.6. Users unable to upgrade should ensure that any downstream application should not allow usage of remotely provided JNDI strings.
CVE-2022-23848	In Alluxio before 2.7.3, the logserver does not validate the input stream. NOTE: this is not the same as the CVE-2021-44228 Log4j vulnerability.
CVE-2022-23307	CVE-2020-9493 identified a deserialization issue that was present in Apache Chainsaw. Prior to Chainsaw V2.0 Chainsaw was a component of Apache Log4j 1.2.x where the same issue exists.
CVE-2022-23305	By design, the JDBCAppender in Log4j 1.2.x accepts an SQL statement as a configuration parameter where the values to be inserted are converters from PatternLayout. The message converter, %m, is likely to always be included. This allows attackers to manipulate the SQL by entering crafted strings into input fields or headers of an application that are logged allowing unintended SQL queries to be executed. Note this issue only affects Log4j 1.x when specifically configured to use the JDBCAppender, which is not the default. Beginning in version 2.0-beta8, the JDBCAppender was re-introduced with proper support for parameterized SQL queries and further customization over the columns written to in logs. Apache Log4j 1.2 reached end of life in August 2015. Users should upgrade to Log4j 2 as it addresses numerous other issues from the previous versions.
CVE-2022-23302	JMSSink in all versions of Log4j 1.x is vulnerable to deserialization of untrusted data when the attacker has write access to the Log4j configuration or if the configuration references an LDAP service the attacker has access to. The attacker can provide a TopicConnectionFactoryBindingName configuration causing JMSSink to perform JNDI requests that result in remote code execution in a similar fashion to CVE-2021-4104. Note this issue only affects Log4j 1.x when specifically configured to use JMSSink, which is not the default. Apache Log4j 1.2 reached end of life in August 2015. Users should upgrade to Log4j 2 as it addresses numerous other issues from the previous versions.
CVE-2022-0070	Incomplete fix for CVE-2021-3100. The Apache Log4j hotpatch package starting with log4j-cve-2021-44228-hotpatch-1.1-16 will now explicitly mimic the Linux capabilities and cgroups of the target Java process that the hotpatch is applied to.
CVE-2021-45105	Apache Log4j2 versions 2.0-alpha1 through 2.16.0 (excluding 2.12.3 and 2.3.1) did not protect from uncontrolled recursion from self-referential lookups. This allows an attacker with control over Thread Context Map data to cause a denial of service when a crafted string is interpreted. This issue was fixed in Log4j 2.17.0, 2.12.3, and 2.3.1.
CVE-2021-45046	It was found that the fix to address CVE-2021-44228 in Apache Log4j 2.15.0 was incomplete in certain non-default configurations. This could allows attackers with control over Thread Context Map (MDC) input data when the logging configuration uses a non-default Pattern Layout with either a Context Lookup (for example, $${ctx:loginId}) or a Thread Context Map pattern (%X, %mdc, or %MDC) to craft malicious input data using a JNDI Lookup pattern resulting in an information leak and remote code execution in some environments and local code execution in all environments. Log4j 2.16.0 (Java 8) and 2.12.2 (Java 7) fix this issue by removing support for message lookup patterns and disabling JNDI functionality by default.
CVE-2021-44832	Apache Log4j2 versions 2.0-beta7 through 2.17.0 (excluding security fix releases 2.3.2 and 2.12.4) are vulnerable to a remote code execution (RCE) attack when a configuration uses a JDBC Appender with a JNDI LDAP data source URI when an attacker has control of the target LDAP server. This issue is fixed by limiting JNDI data source names to the java protocol in Log4j2 versions 2.17.1, 2.12.4, and 2.3.2.
CVE-2021-44530	An injection vulnerability exists in a third-party library used in UniFi Network Version 6.5.53 and earlier (Log4J CVE-2021-44228) allows a malicious actor to control the application.
CVE-2021-44228	Apache Log4j2 2.0-beta9 through 2.15.0 (excluding security releases 2.12.2, 2.12.3, and 2.3.1) JNDI features used in configuration, log messages, and parameters do not protect against attacker controlled LDAP and other JNDI related endpoints. An attacker who can control log messages or log message parameters can execute arbitrary code loaded from LDAP servers when message lookup substitution is enabled. From log4j 2.15.0, this behavior has been disabled by default. From version 2.16.0 (along with 2.12.2, 2.12.3, and 2.3.1), this functionality has been completely removed. Note that this vulnerability is specific to log4j-core and does not affect log4net, log4cxx, or other Apache Logging Services projects.
CVE-2021-4125	It was found that the original fix for log4j CVE-2021-44228 and CVE-2021-45046 in the OpenShift metering hive containers was incomplete, as not all JndiLookup.class files were removed. This CVE only applies to the OpenShift Metering hive container images, shipped in OpenShift 4.8, 4.7 and 4.6.
CVE-2021-4104	JMSAppender in Log4j 1.2 is vulnerable to deserialization of untrusted data when the attacker has write access to the Log4j configuration. The attacker can provide TopicBindingName and TopicConnectionFactoryBindingName configurations causing JMSAppender to perform JNDI requests that result in remote code execution in a similar fashion to CVE-2021-44228. Note this issue only affects Log4j 1.2 when specifically configured to use JMSAppender, which is not the default. Apache Log4j 1.2 reached end of life in August 2015. Users should upgrade to Log4j 2 as it addresses numerous other issues from the previous versions.
CVE-2021-3100	The Apache Log4j hotpatch package before log4j-cve-2021-44228-hotpatch-1.1-13 didn’t mimic the permissions of the JVM being patched, allowing it to escalate privileges.
CVE-2020-9493	A deserialization flaw was found in Apache Chainsaw versions prior to 2.1.0 which could lead to malicious code execution.
CVE-2020-9488	Improper validation of certificate with host mismatch in Apache Log4j SMTP appender. This could allow an SMTPS connection to be intercepted by a man-in-the-middle attack which could leak any log messages sent through that appender. Fixed in Apache Log4j 2.12.3 and 2.13.1
CVE-2019-3826	A stored, DOM based, cross-site scripting (XSS) flaw was found in Prometheus before version 2.7.1. An attacker could exploit this by convincing an authenticated user to visit a crafted URL on a Prometheus server, allowing for the execution and persistent storage of arbitrary scripts.
CVE-2019-17571	Included in Log4j 1.2 is a SocketServer class that is vulnerable to deserialization of untrusted data which can be exploited to remotely execute arbitrary code when combined with a deserialization gadget when listening to untrusted network traffic for log data. This affects Log4j versions up to 1.2 up to 1.2.17.
CVE-2019-17531	A Polymorphic Typing issue was discovered in FasterXML jackson-databind 2.0.0 through 2.9.10. When Default Typing is enabled (either globally or for a specific property) for an externally exposed JSON endpoint and the service has the apache-log4j-extra (version 1.2.x) jar in the classpath, and an attacker can provide a JNDI service to access, it is possible to make the service execute a malicious payload.
CVE-2017-5645	In Apache Log4j 2.x before 2.8.2, when using the TCP socket server or UDP socket server to receive serialized log events from another application, a specially crafted binary payload can be sent that, when deserialized, can execute arbitrary code.

https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=log4j

A.3d Weaknesses: Tools

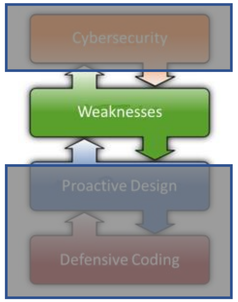
MITRE: from Cold-War era



"MITRE began in 1958, sponsored by the U.S. Air Force to bridge across the academic research community and industry to architect the Semi-Automatic Ground Environment, or SAGE, a key component of Cold War-era air defense. We were founded as a not-for-profit company to serve as objective advisers in systems engineering to government agencies, both military and civilian.
We are innovators—from advances in radar technology, cyber, GPS, cancer research, and aviation collision-avoidance systems to breakthroughs in evolving disciplines such as vehicle autonomy, artificial intelligence, and synthetic biology.
Moreover, as a company that doesn't compete with industry, we're uniquely positioned to convene government, industry, and academia to collaborate on big societal challenges, from pandemic response to highway safety to social justice.
At its core, MITRE's story is about our people. We're proud that more than 9,000 multi-talented and creative individuals choose to stand with us every day, dedicating themselves to our mission of solving problems for a safer world."

A.3d Weaknesses: Tools

MITRE: federal research

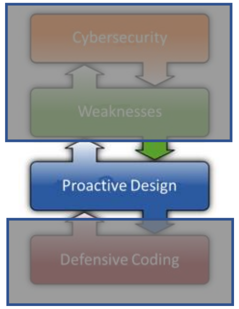


(Interestingly, MITRE is not an acronym, though some thought it stood for Massachusetts Institute of Technology Research and Engineering. The name is the creation of James McCormack, an early board member, who wanted a name that meant nothing, but sounded evocative.)

“We discover. We create. We lead.
MITRE is trusted to lead–by government, industry, and academia.
The bedrock of any trusted relationship is integrity. For more than 60 years, MITRE has proudly operated federally funded research and development centers, or FFRDCs. We now operate six of the 42 FFRDCs in existence—a high honor.
Since our inception, MITRE has consistently addressed the most complex whole-of-nation challenges that threaten our country’s safety, security, and prosperity. Our mission-driven teams bring technical expertise, objectivity, and an interdisciplinary approach to drive innovation and accelerate solutions in the public interest.
Above all, MITRE is trusted to deliver data-driven results and recommendations without any conflicts of interest.”

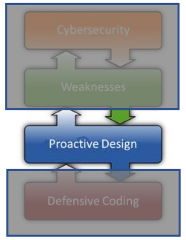
A.4 Proactive Design: Best Practices, Architecture, Processes

Useful Lists of Well-done Actions for Secure Implementation



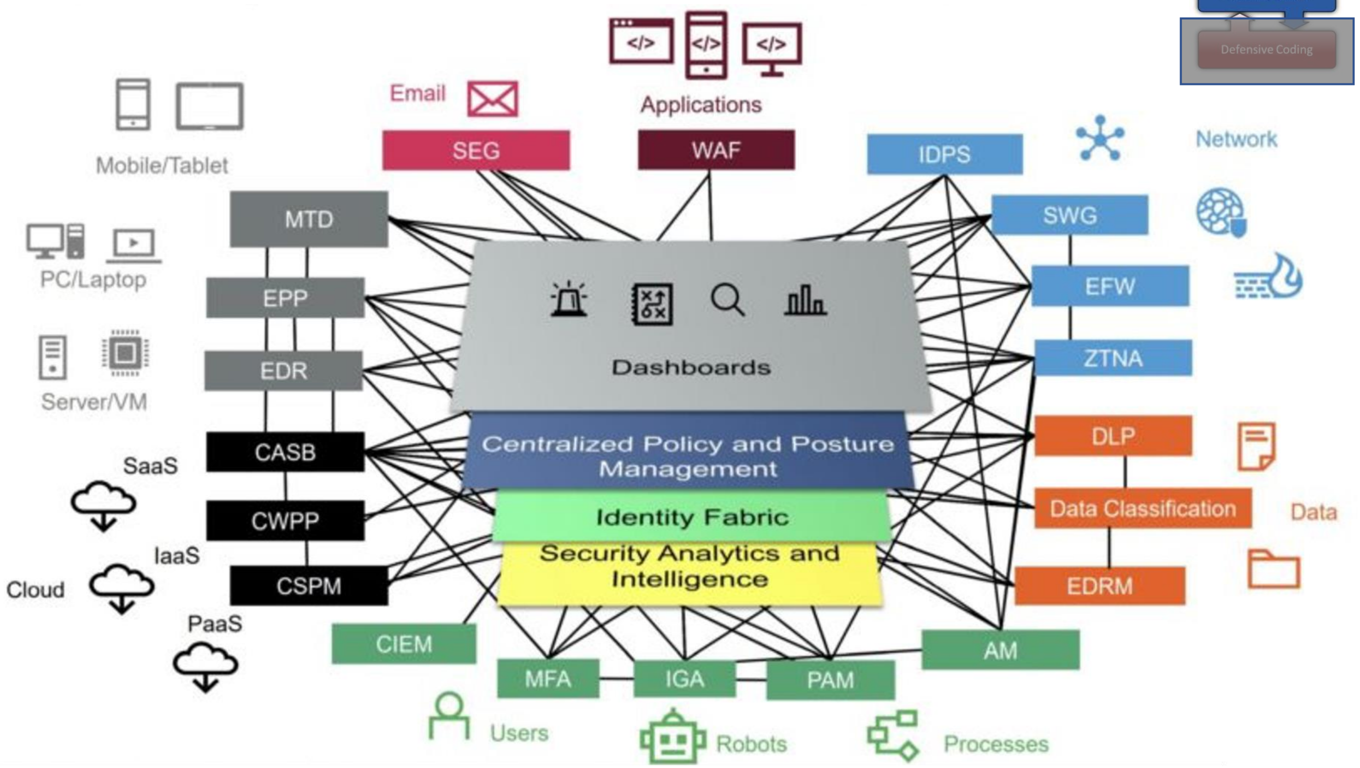
A.1d Secure Programming: Introduction

Proactive Design (where): safer architecture integration



Nowadays application software should guarantee interoperability, that is the ability to communicate and share information about cybersecurity.

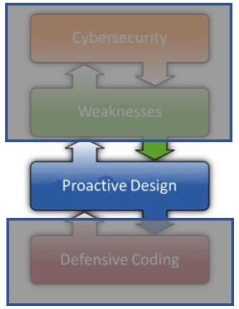
No more silos: every component is part of a bigger infrastructure, giving some service and obtaining some other back.



Gartner CSMA: Cyber Security Mesh Architecture

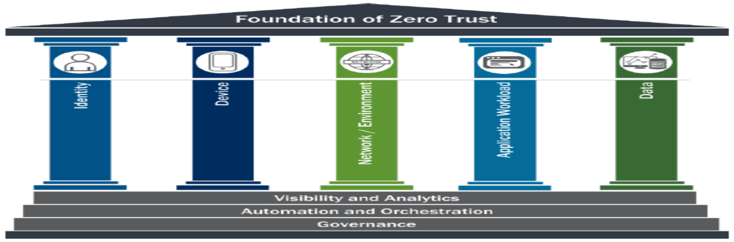
A.4 Proactive Design: Best Practices, Architecture, Processes

Useful Lists of Well-done Actions for Secure Implementation



1. **NIST CSF**: National Cybersecurity Framework (focused in **How-To** manage an **incident**)

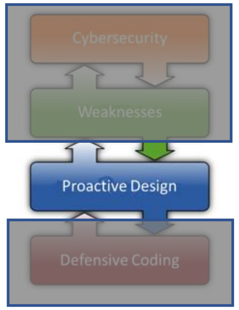
2. **ZTA**: Zero Trust Architecture («**Never Trust**, **Always Verify**»)

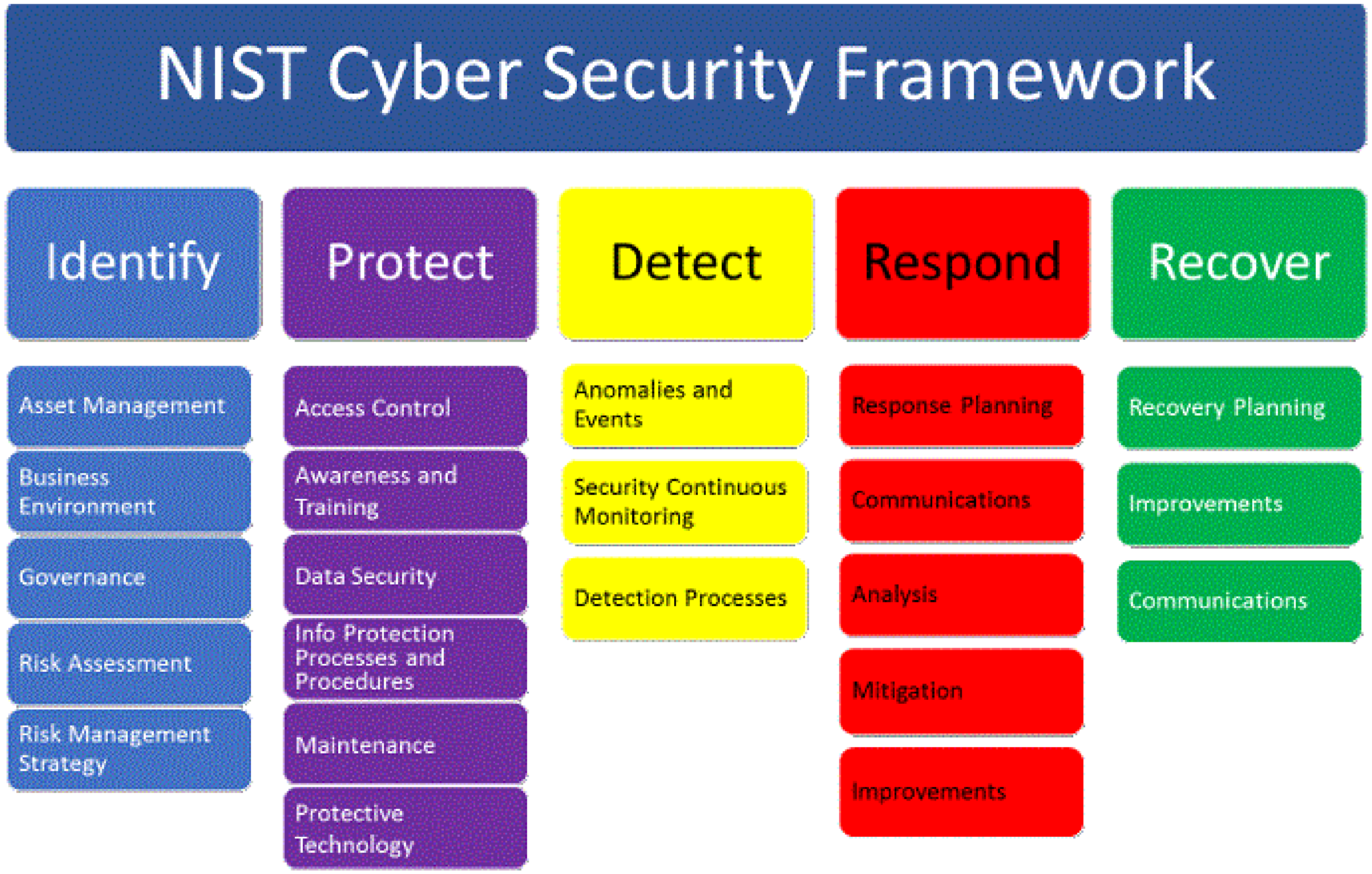


3. **DevSecOps**: **Shift Left** (not ~~Implementing Security~~ but **Securing Implementation**)

A.4a Proactive Design: Best Practice

NIST Cyber Security Framework





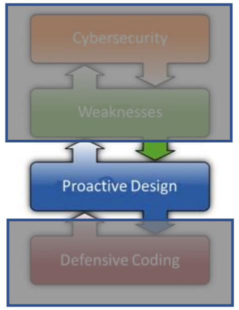
The five Functions and their subcategories of NIST CSF

The NIST Cybersecurity Framework (CSF) is a risk-based approach designed for businesses to assess and manage cybersecurity risk.

Although the framework is published by the United States Department of Commerce agency, the common taxonomy of standards, guidelines, and practices that it provides is not country-specific; this explains why it is used by many governments, businesses, and organizations worldwide.

A.4.b Proactive Design: Best Practice

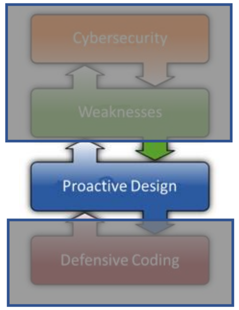
NIST Cybersecurity Framework: Functions and Categories

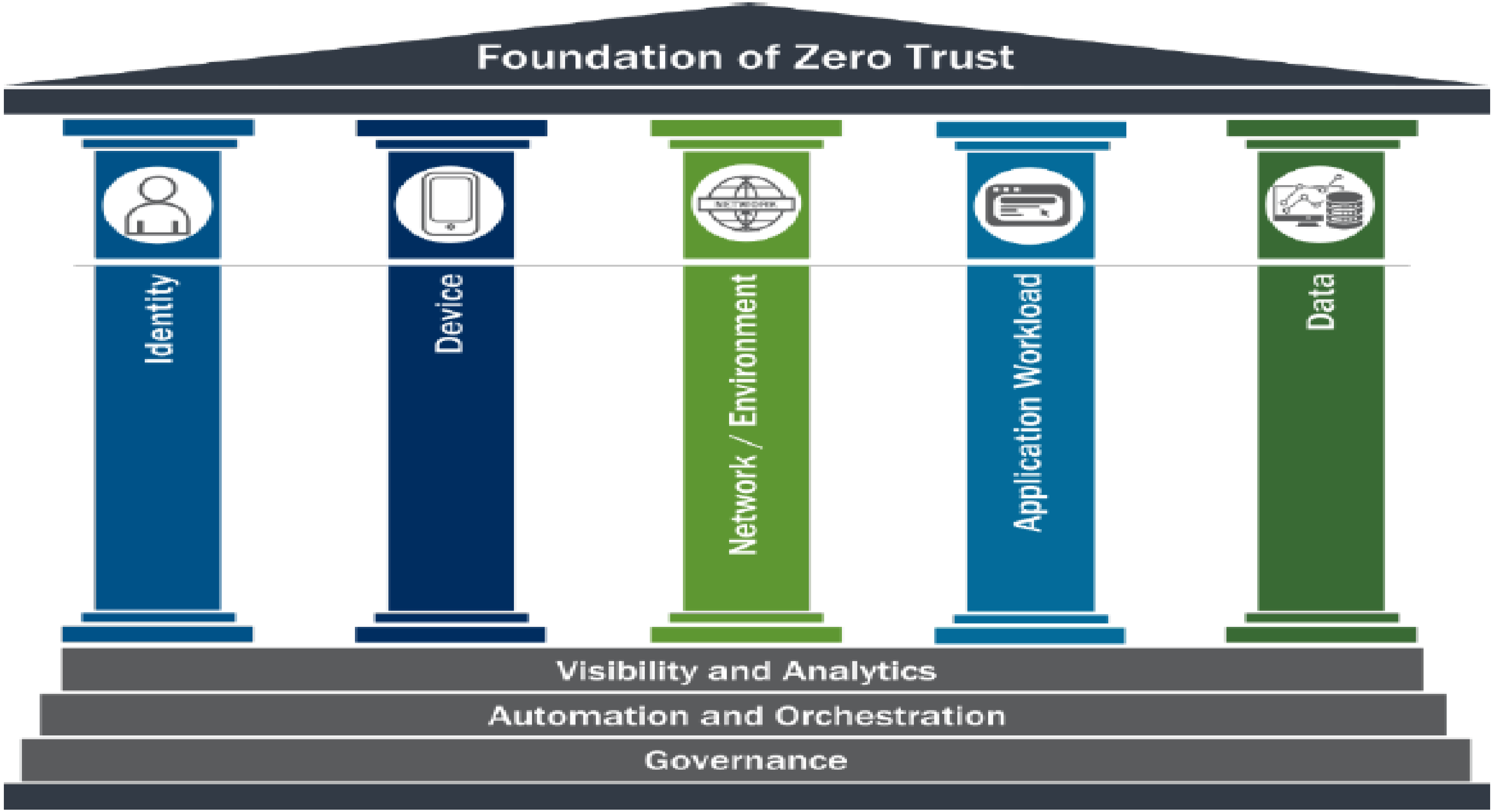


Function Identifier	Function	Category Identifier	Category
ID	Identify	ID.AM	Asset Management
		ID.BE	Business Environment
		ID.GV	Governance
		ID.RA	Risk Assessment
		ID.RM	Risk Management Strategy
		ID.SC	Supply Chain Risk Management
PR	Protect	PR.AC	Identity Management and Access Control
		PR.AT	Awareness and Training
		PR.DS	Data Security
		PR.IP	Information Protection Processes and Procedures
		PR.MA	Maintenance
		PR.PT	Protective Technology
DE	Detect	DE.AE	Anomalies and Events
		DE.CM	Security Continuous Monitoring
		DE.DP	Detection Processes
RS	Respond	RS.RP	Response Planning
		RS.CO	Communications
		RS.AN	Analysis
		RS.MI	Mitigation
		RS.IM	Improvements
RC	Recover	RC.RP	Recovery Planning
		RC.IM	Improvements
		RC.CO	Communications

A.4.c Proactive Design: Architecture

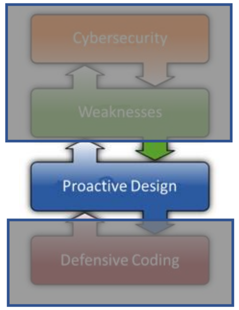
ZTA: Zero Trust Architecture

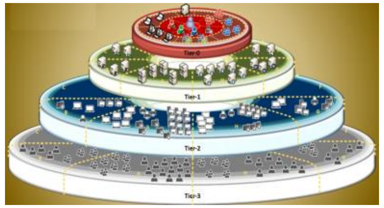
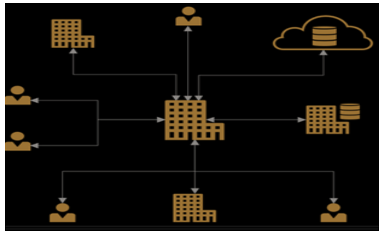




A.4.d Proactive Design: Architecture

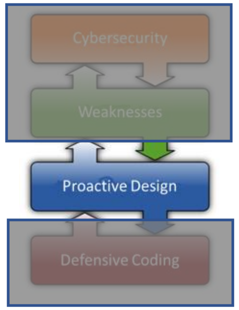
ZTA: Evolution of Trust Models & Topologies



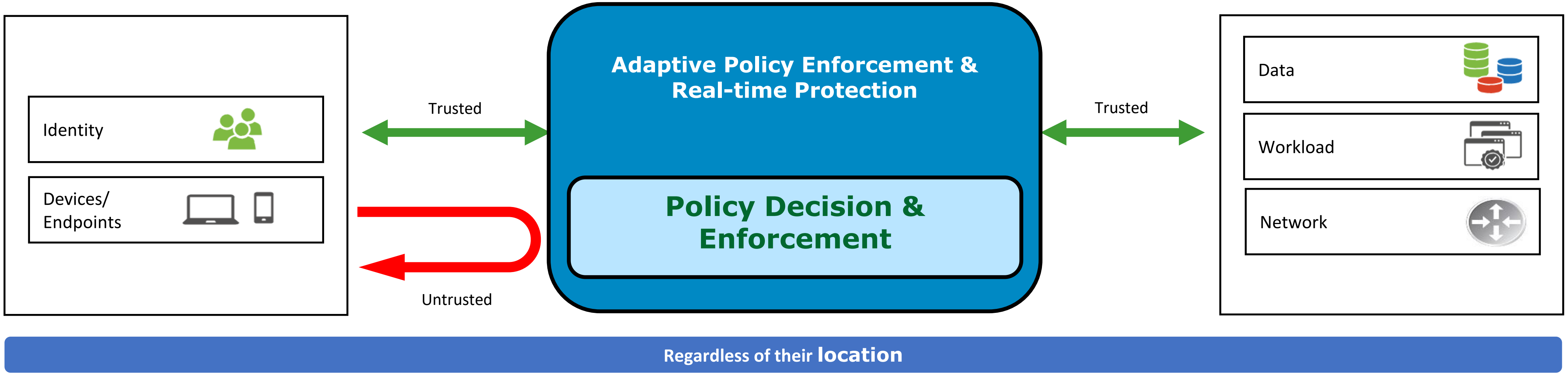
	Years	Name	Fashion	Remote	Description	Trust	Tools	Drawback
	‘90s	**Tier Model** strict separation of assets	«Circles of Hell»	No / a Few	logical separation of assets by boundaries in the same physical location (old-fashioned **Perimeter-Centric**).	Inside Yes, Outside. No	FW IDS	No Remote
	’00s	**Hub & Spoke** connect outlying points to a central "hub".	«Airline Routes»	Some	remote connections secured by VPN tunnels (strong pub-key cryptography) converging at one location (**Centralized Branch Office**)	Outside could get as Inside	VPN SSL-VPN VDI RDP	Bottleneck and SPoF
	‘20s	**Zero Trust** Authentication GW Distribution	«Never Trust, Always Verify»	Most	connections are granted after careful verification (Identity, Device, Time, Geolocation, Security Posture (**Default Deny**)	per-transaction basis.	PEP (CASB, ATP, DLP, …) ➔SASE	Distributed network of PoPs

A.4.e Proactive Design: Architecture

What is ZTA



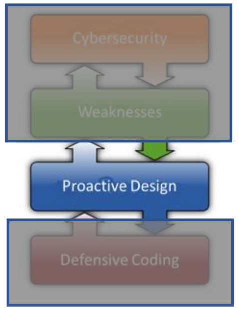
Alternative Cybersecurity model,

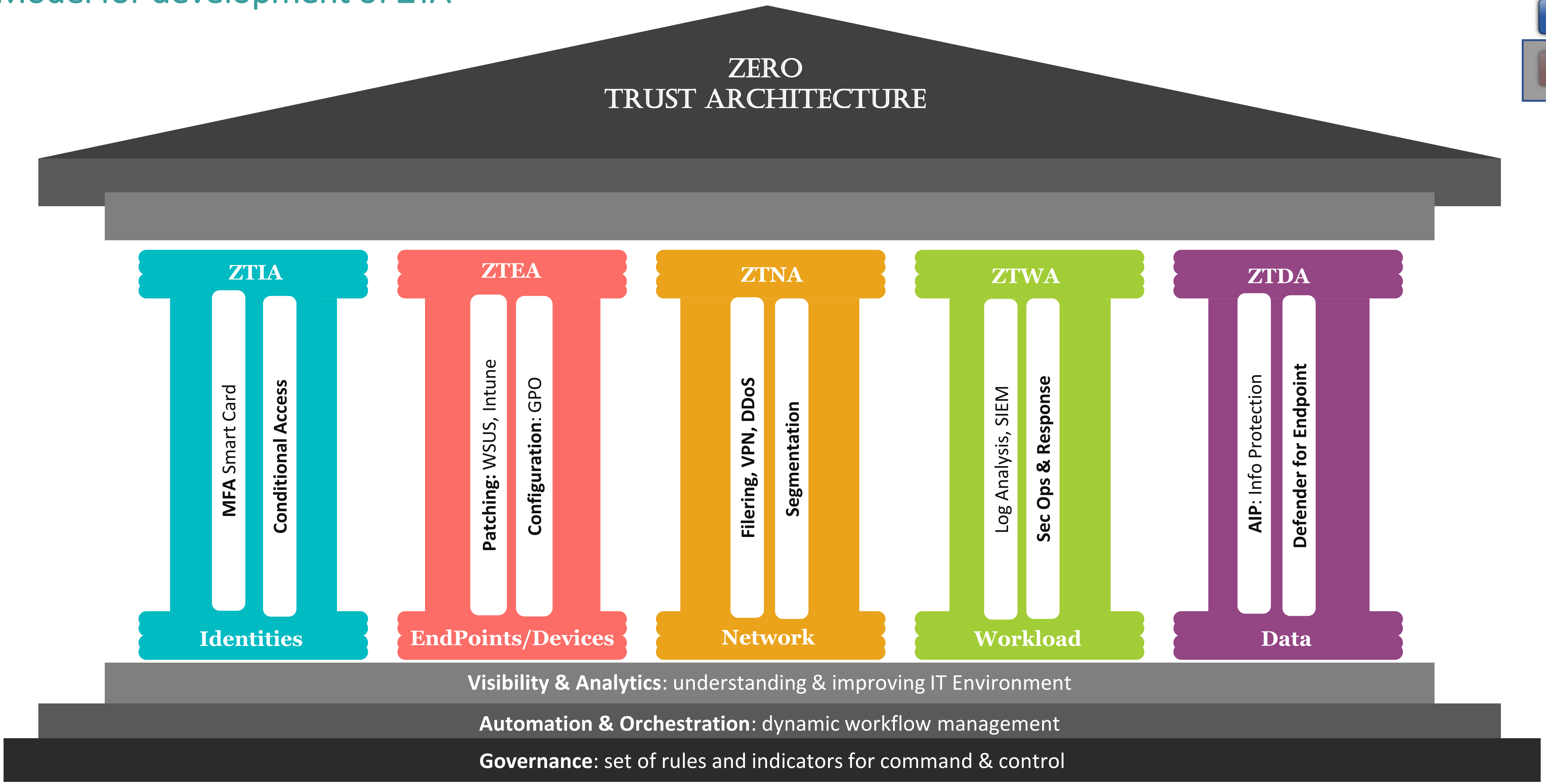


- Focusing on Protecting Data rather than access to devices, removing the assumption of perimeter trust.
- Enforcing Access Control by a Decision/Enforcement Point, based not more only on Network rules but on dynamic Policies calculated on continuous verification
- Assuming Identity as the new front line (together with accessing device), continuously assessing it and his behaviours.

A.4.f Proactive Design: Architecture

Pillar Model for development of ZTA

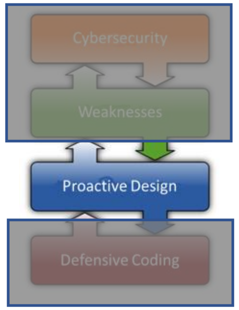


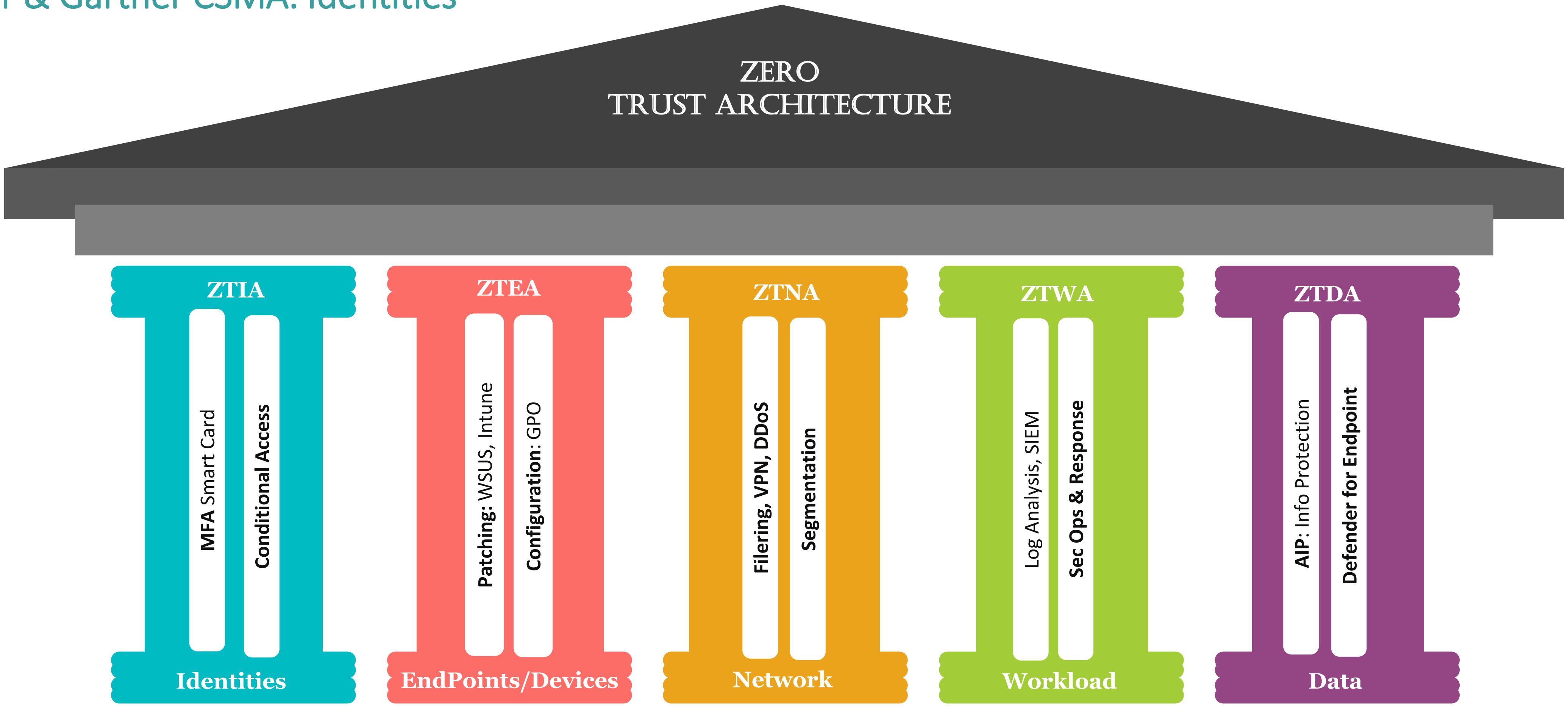


This model cames from CISA [ZT Maturity Model]

A.4.f1 Proactive Design: Architecture

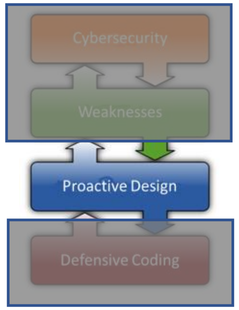
ZTA Pillar & Gartner CSMA: Identities

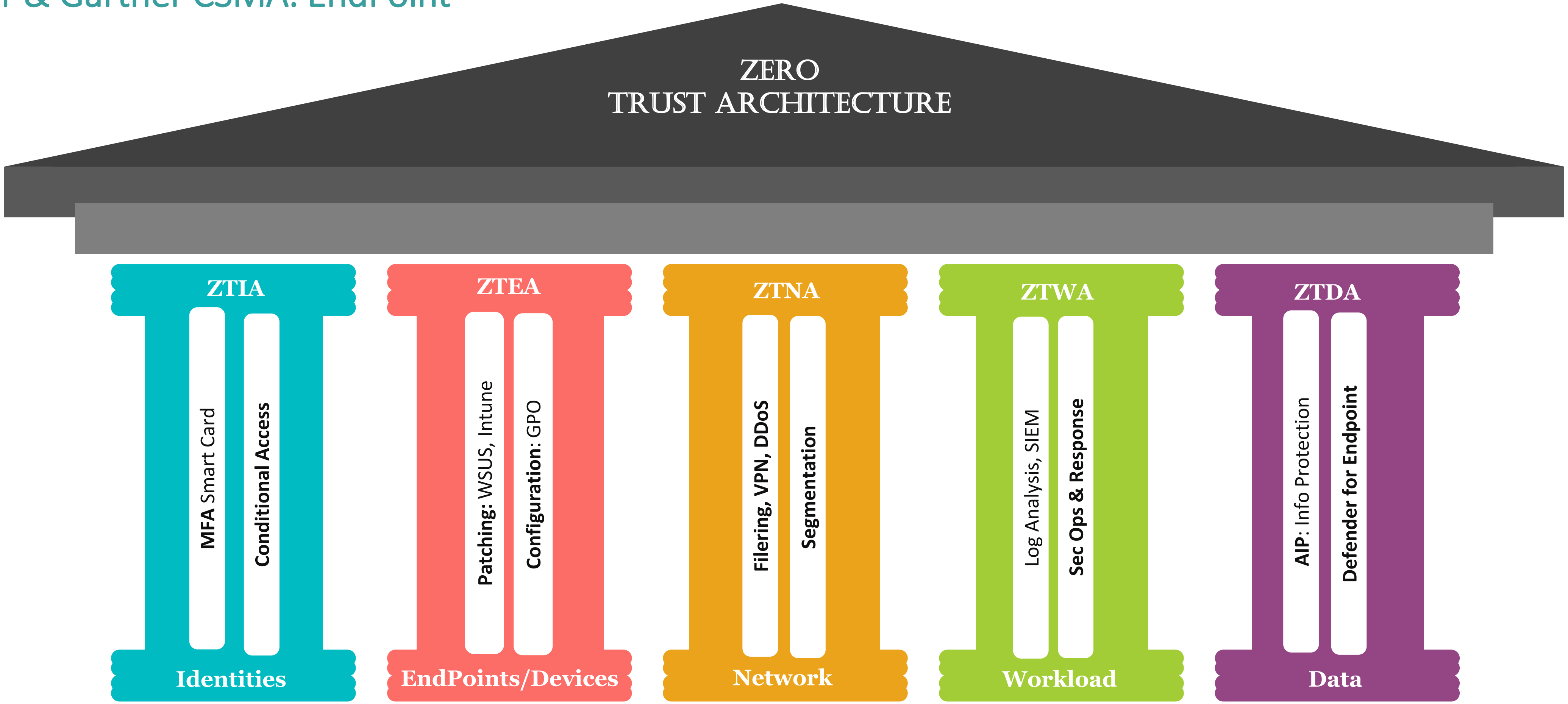


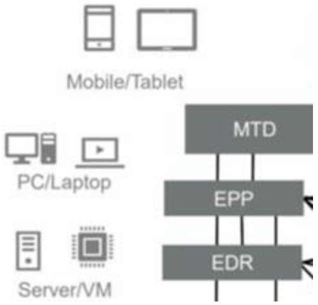


A.4.f2 Proactive Design: Architecture

ZTA Pillar & Gartner CSMA: EndPoint

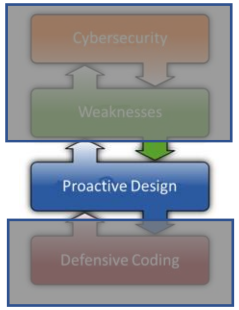


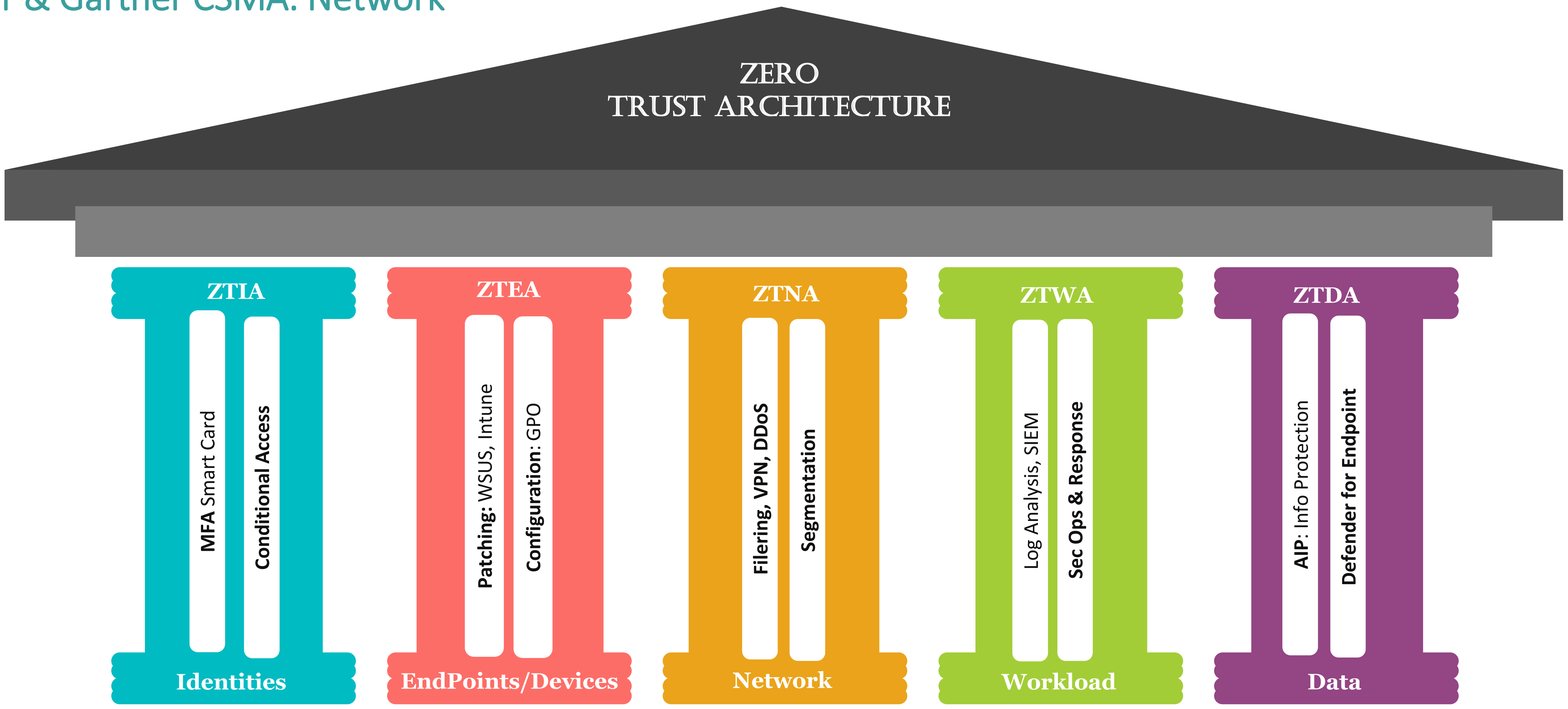


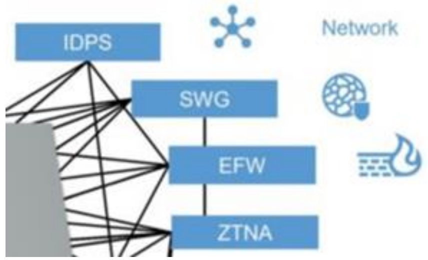


A.4.f3 Proactive Design: Architecture

ZTA Pillar & Gartner CSMA: Network

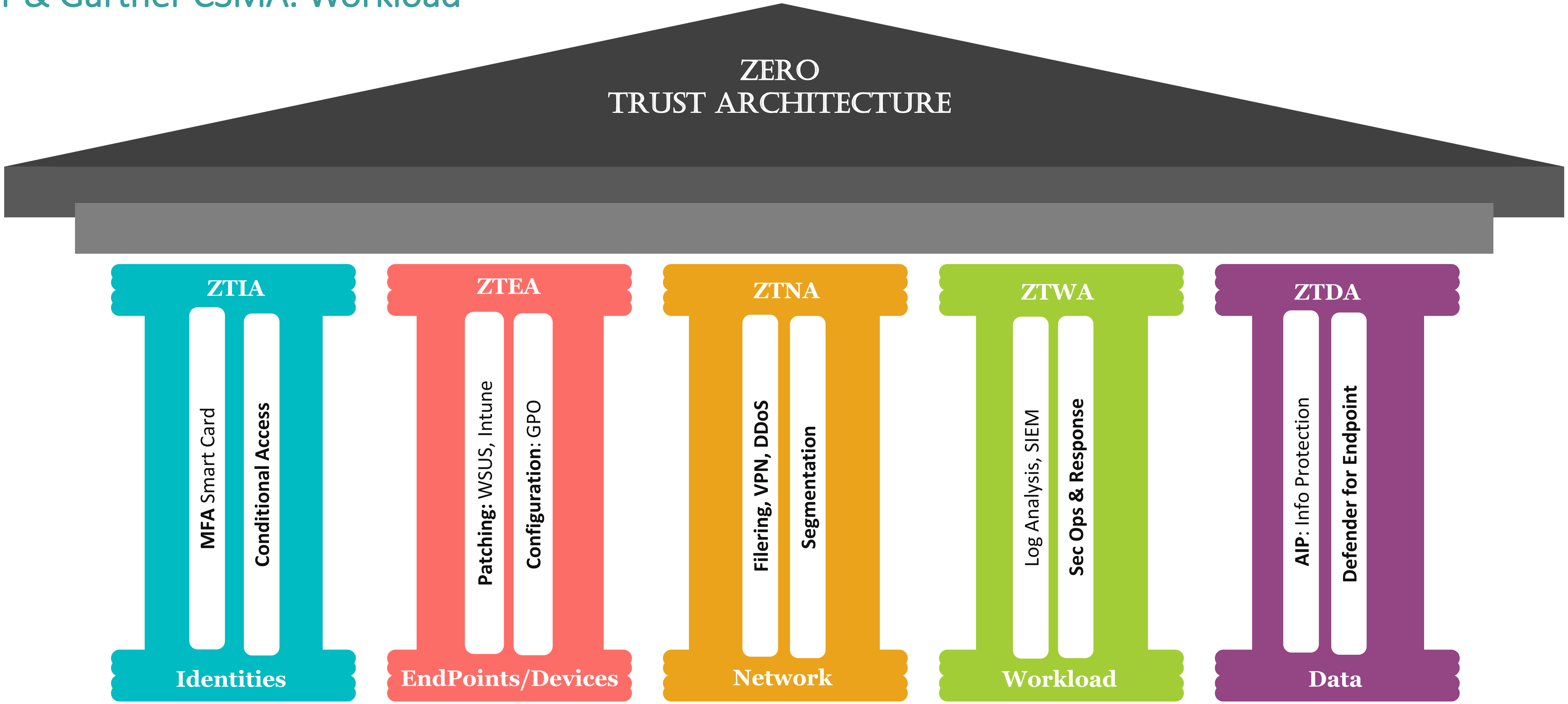


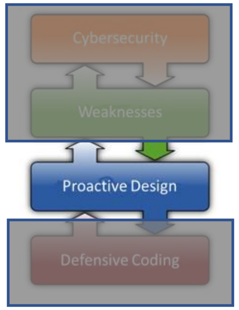




A.4.f4 Proactive Design: Architecture

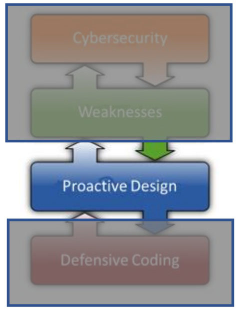
ZTA Pillar & Gartner CSMA: Workload

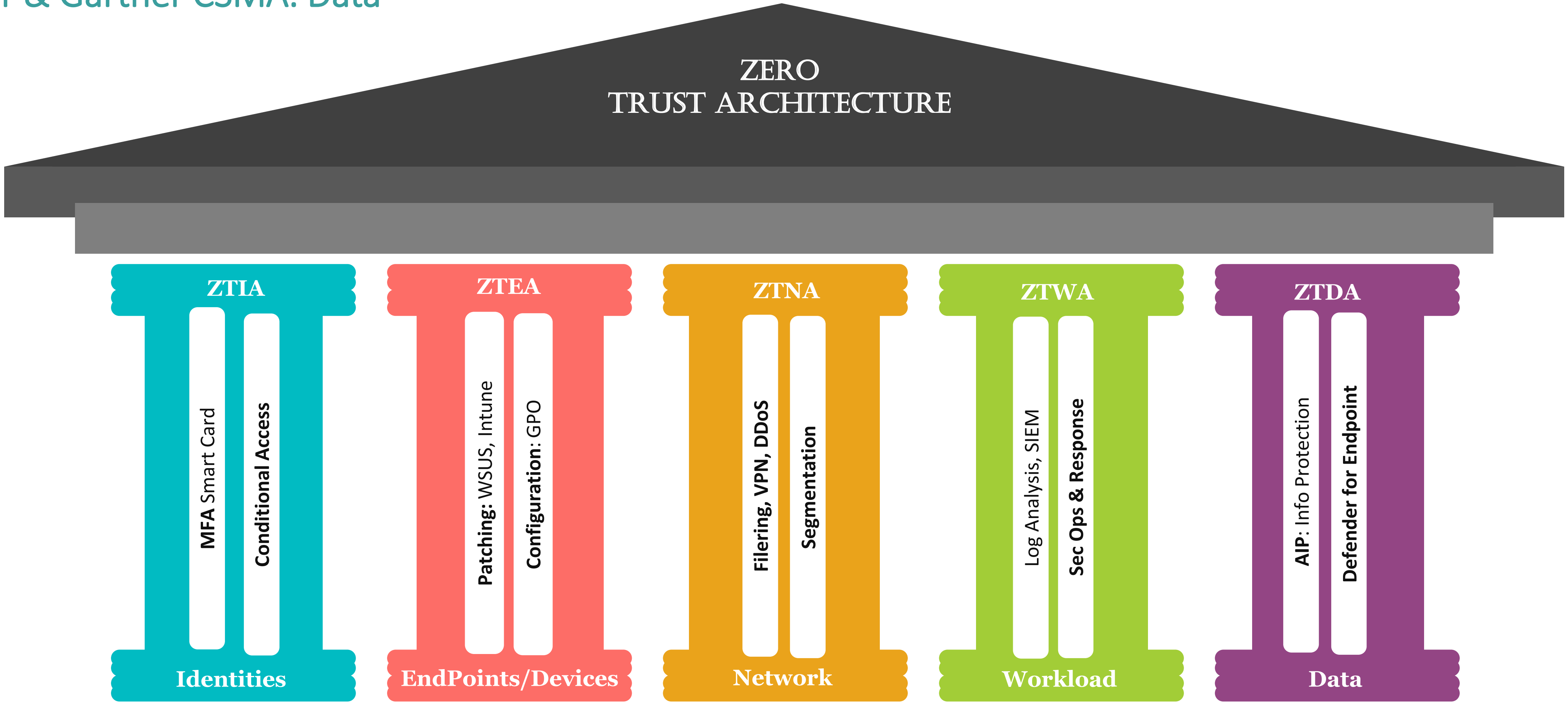




A.4.f5 Proactive Design: Architecture

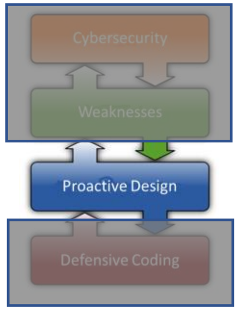
ZTA Pillar & Gartner CSMA: Data

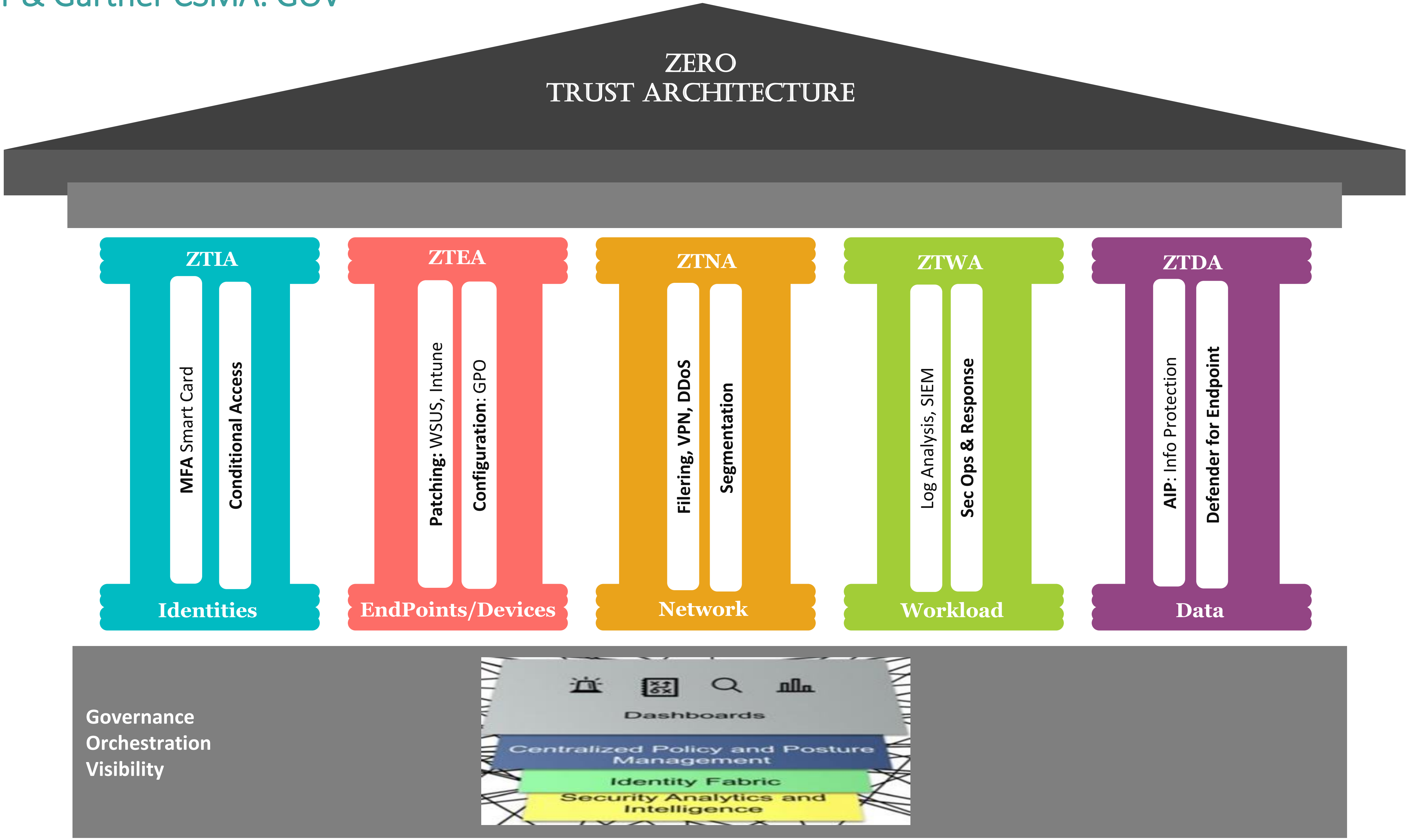




A.4.f6 Proactive Design: Architecture

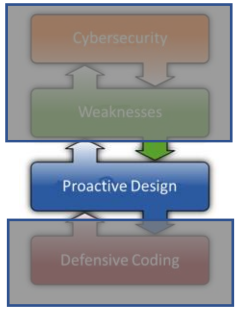
ZTA Pillar & Gartner CSMA: GOV





A.4.g Proactive Design: Architecture

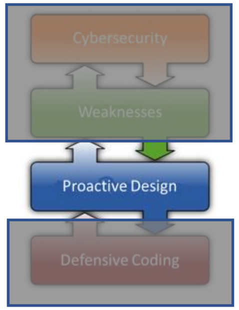
NIST Cyber Security Framework: Category mapping for Pervasive Telemetry



Categories

Identify	Asset Management ID.AM	Business Environment ID.BE	Risk Assessment ID.RA	Risk Management Strategy ID.RM	Supply Chain Risk Management ID.SC	
Protect	Identity Management, Authentication, and Access Control PR.AC	Awareness and Training PR.AT	Data Security PR.DS	Information Protection Processes and Procedures PR.IP	Maintenance PR.MA	Protective Technology PR.PT
Detect	Anomalies and Events DE.AE	Security Continuous Monitoring DE.CM	Detection Processes DE.DP			
Respond	Response Planning RC.RP	Communications RC.CO	Analysis RS.AN	Mitigation RS.MI	Improvements RS.MI	
Recover	Recovery Planning RC.RP	Improvements RC.IM	Communications RC.CO			

As mapped by NCCoE in the paper "[Implementing a ZTA]"

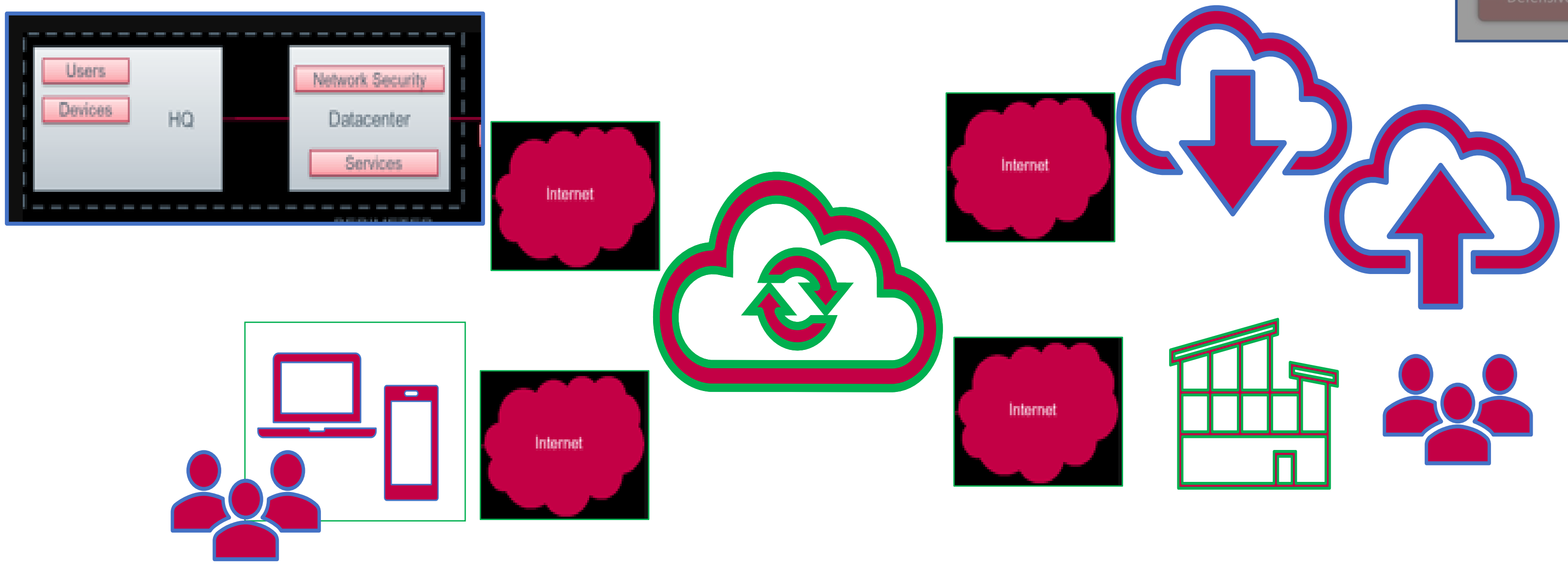


A.4.h Proactive Design: Architecture

ZTA: IT Functions: security & protection

Several tools enabling ZTA for Hybrid Cloud.

Those could be classified on:

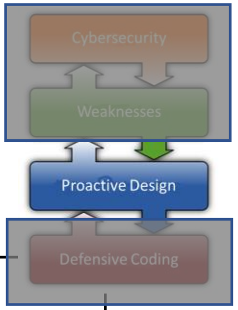


- **Infrastructure**: tools for security management of the Hybrid Cloud components, its usage readiness and configuration. That is, by *static* point of view, focused on the management of the service items and their status. Without direct relation to any specific connection, interaction, activity (about 2/3 of the tools).

- **Transaction**: tools for security & management of any specific connection, interaction, activity amidst the Hybrid Cloud. That is, by *dynamic* point of view, focused on access, about the usage of the configuration set by the infrastructure tool (about 1/3 of the tools). Often integrated in **SASE** platforms and **SD-WAN** as well.

A.4.i Proactive Design: Architecture

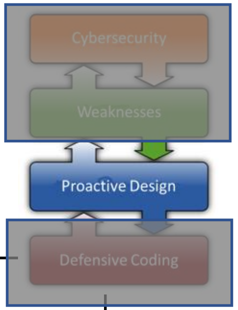
ZTA: Platforms for protecting infrastructure



Pillar(s)	Function	Name	Enforce	Enabling
Identity	**IGA**	Identity Governance (SoD)	Authorizations: Permissions	Identity Lifecycle.
Identity	**CIEM**	Cloud Infrastructure Entitlement Management	Roles: Entitlements	Business & Application Lifecycle
Identity	**PAM**	Privileged Access Management	Authorizations: Privileged	Privilege Administration
EndPoint	**CMDB**	Asset Mgmt	Item identification	Item Configuration
EndPoint	**MDM**	Mobile Device Management	Patching	Vulnerability Management; Change & Configuration Mgmt
Network	**CNS**	Cloud Network Security	Segregation & Segmentation	Micro-Segmentation
Network Workload	**DDoS**	Anti-DDoS	Protect against obscuration	Application Availability
Workload	**SCM**	SW Configuration Mgmt	Config & Change	Approval Workflow
Workload	**CSPM**	Cloud Security Posture Mgmt	Secure Configuration	Compliance
Workload	**CWP**	Cloud Workload Protection	SW Mgmt	Configuration Management
Workload	**XDR**	eXtended Detection & Response	Threat Detection	Block advanced malware, exploits and fileless attacks
Workload	**IRM**	Integrated Risk Management	Security Dashboard	Security Governance by KPI
Data	**CKMS**	Cloud Key Mgmt Service	Secure Key Mgmt	Centralized key control in hybrid cloud

A.4.i Proactive Design: Architecture

ZTA: Platforms for Protecting Transaction ➔ SASE

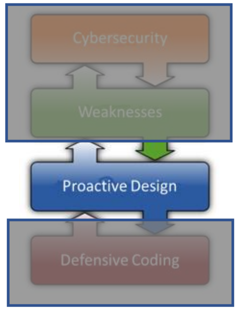


Pillar(s)	Function	Name	Enforce	Enabling
Identity Workload	**CASB**	Cloud Access Security Broker	threats, and data leakage identification	Access to cloud applications and shadow IT
EndPoint	**SWG**	Secure Web Gateway	URL filtering	Access to Internet
EndPoint	**ATP**	Advanced Threat Prevention	Blocking threats	Spreading across endpoints and nets.
EndPoint Network	**DNS-Sec**	DNS Security	predicting, blocking, and tracking malicious activity	Access to Internet
Network	**VPN**	Virtual Private Network	threats, and data leakage	Access to shadow IT
Network	**SD-WAN**	SW Defined WAN	intelligent unified view and simplified mgmt	Traffic Prioritization, WAN Optimization, converged backbones)
Network	**FWaaS**	FW as a Service	Next Generation Rules	Net Filtering
Data	**DLP**		Detecting/Blocking Exfiltration	Access to Company Data

Not all SASE vendors do implement all the listed ZTA functions

A.4f Proactive Design: Processes

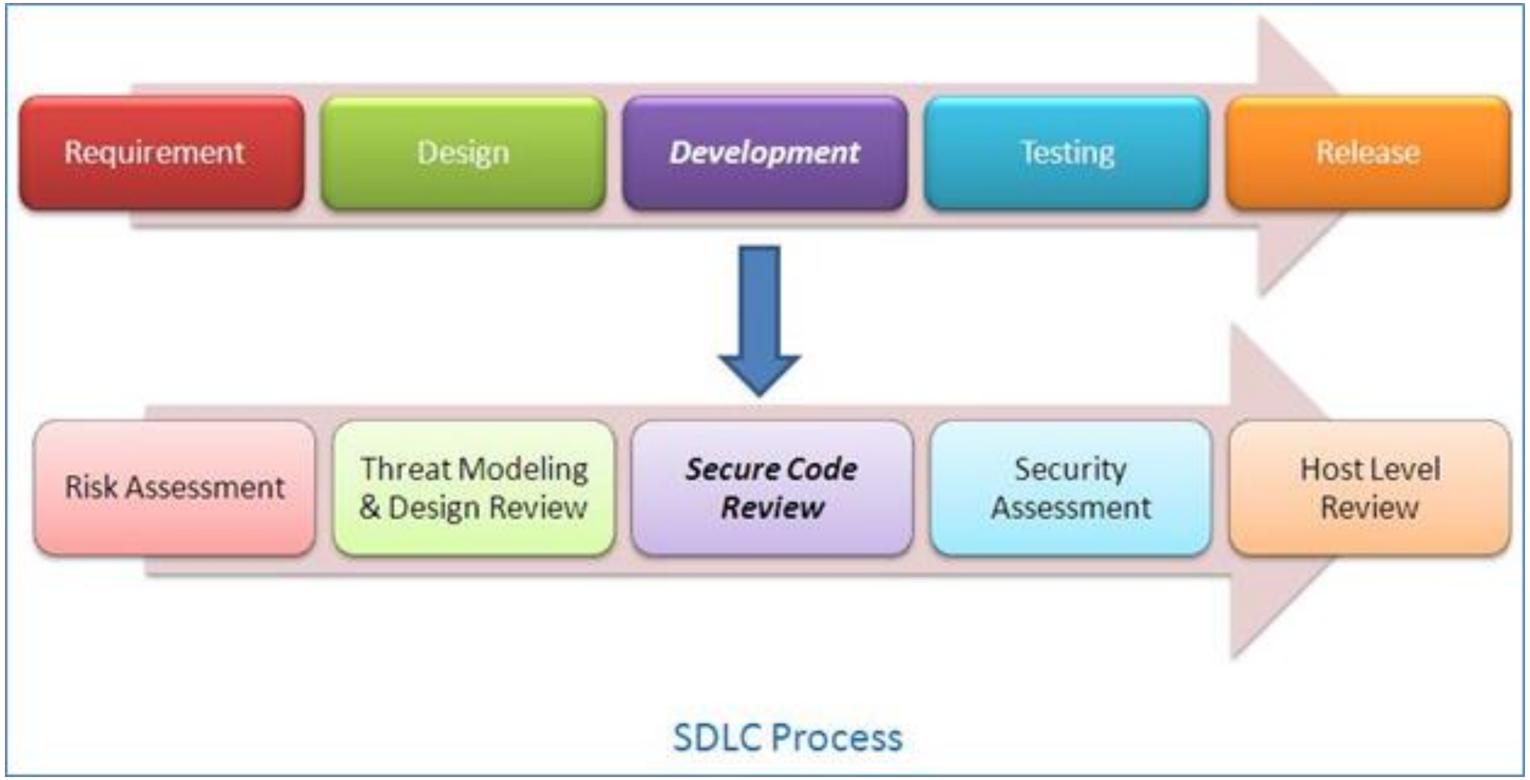
SDLC and Security: DevSecOps



Secure Code Review is a process which identifies the insecure piece of code which may cause a potential vulnerability in a later stage of the software development process, ultimately leading to an insecure application.

When a vulnerability is detected in earlier stages of SDLC, it has less impact than the later stages of SDLC – when the insecure code moves to the production environment.

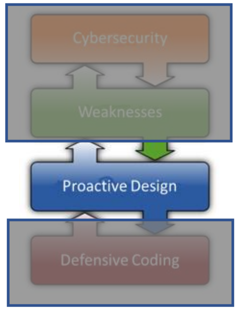
In the SDLC, the secure code review process comes under the Development Phase, which means that when the application is being coded by the developers, they can do self-code review or a security analyst can perform the code review, or both.



Software Development Life Cycle and Security

A.4f Proactive Design: Processes

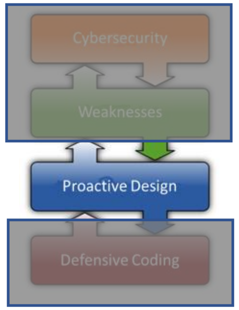
DevSecOps: Shift Left Approach



Shift Left is **a practice intended to find and prevent defects early in the software delivery process**. The idea is to improve quality by moving tasks to the left as early in the lifecycle as possible. Shift Left testing means testing earlier in the software development process.

A.4g Proactive Design: Processes

DevSecOps: Shift Left Approach



The Technology Driving Shift Left Security

DevOps organizations realized that they must also shift security left to avoid introducing more security risks than security and operations teams can manage. This movement is known as DevSecOps, and uses a variety of tools and technologies to close the gap and enable rapid, automated security assessment as part of the CI/CD pipeline:

•**Static Application Security Testing (SAST)** is used to scan source code for known weaknesses and insecure coding practices. In DevSecOps, this testing is typically integrated into developers' development environments for immediate security risk feedback.

•**Software Composition Analysis (SCA)** analyzes software to detect known software components, such as open source and third-party libraries, and identify any associated vulnerabilities. SCA complements SAST by finding vulnerabilities not detectable by scanning source code.

•**Dynamic Application Security Testing (DAST)** scans applications in runtime, prior to deployment into production environments. This enables an outside-in approach to testing applications for exploitable conditions that were not detectable in a static state.

•**Runtime Application Self-Protection (RASP)** runs alongside applications in production to observe and analyze behavior and notify or block anomalous and unauthorized actions. While this may place additional infrastructural burden on production environments, it delivers a real-time look into potential application security risks.

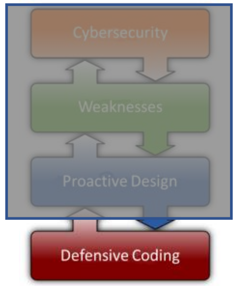
•**Web Application Firewalls (WAF)** monitor traffic at the application level and detect potential attacks and attempts to exploit vulnerabilities. WAFs can be configured to block certain potential attack vectors even without remediating the underlying software vulnerabilities.

•**Container image scanning tools** can continuously and automatically scan container images within the CI/CD pipeline and in container registries, prior to deployment into production environments. This enables identification of vulnerabilities or unsafe components, and provides remediation or mitigation guidance directly to developers and DevOps teams.

•**Cloud Security Posture Management (CSPM)** solutions identify misconfigurations in cloud infrastructure that could leave potential risks and attack vectors unchecked. CSPM solutions can recommend or automatically apply security best practices based on an organization's internal policies or third-party security standards.

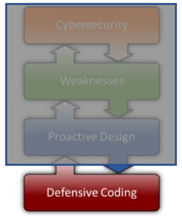
A.5 Code Vulnerability: Security Bugs

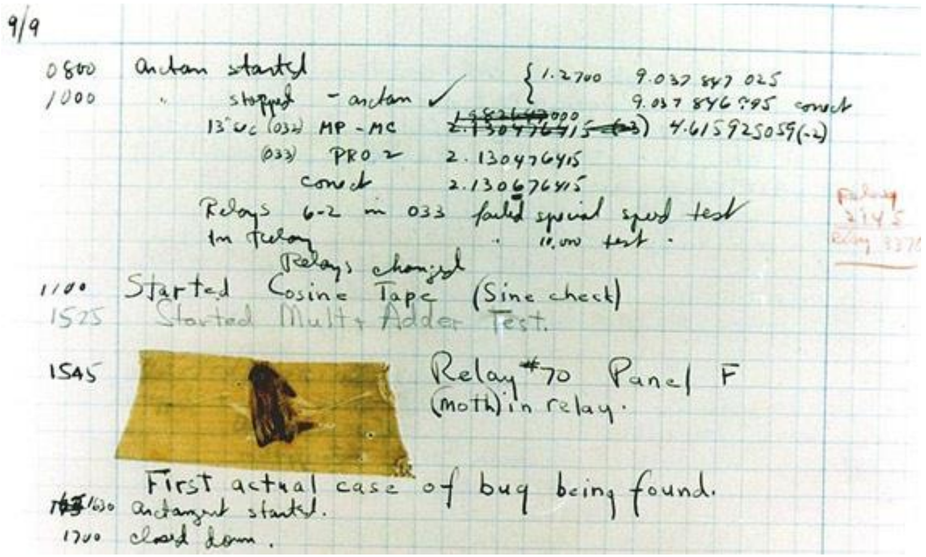
Definition



A.1e Secure Programming: Introduction

Defensive Coding (how): developing without security bugs





The first bug (Source: Naval Historical Center Online Library Photograph)

The causes of security breaches are varied, but many of them owe to a defect (or **bug**) or design flaw in a targeted computer system's software.

After finding a moth inside the Harvard Mark II computer on September 9th, 1947 at 3:45 p.m., Grace Murray Hopper logged the first computer bug in her log book.

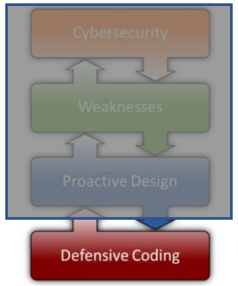
She wrote the time and the sentence: “First actual case of bug being found”.

Nowadays, the term “bug” in computer science is not taken literally, of course. We use it to talk about a flaw or failure in a computer program that causes it to produce an unexpected result or crash.

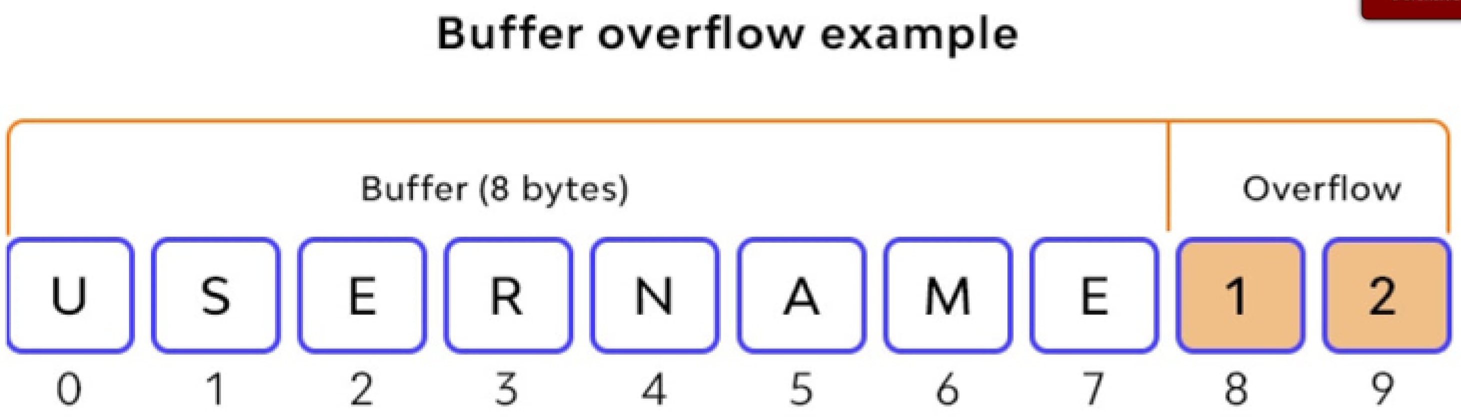
Buffers contain a certain amount of data that limits it to hold limited data for a limited time as multiple

A.5 Code Vulnerability: Buffer Overflow

Definition



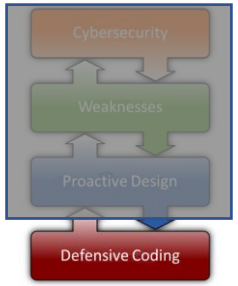
Buffers contain a certain amount of data that limits it to hold limited data for a limited time as multiple application uses this mechanism of the buffer. Resultantly a situation arrives when further data is pushed into a buffer, such a condition refers to a term called a buffer overflow.

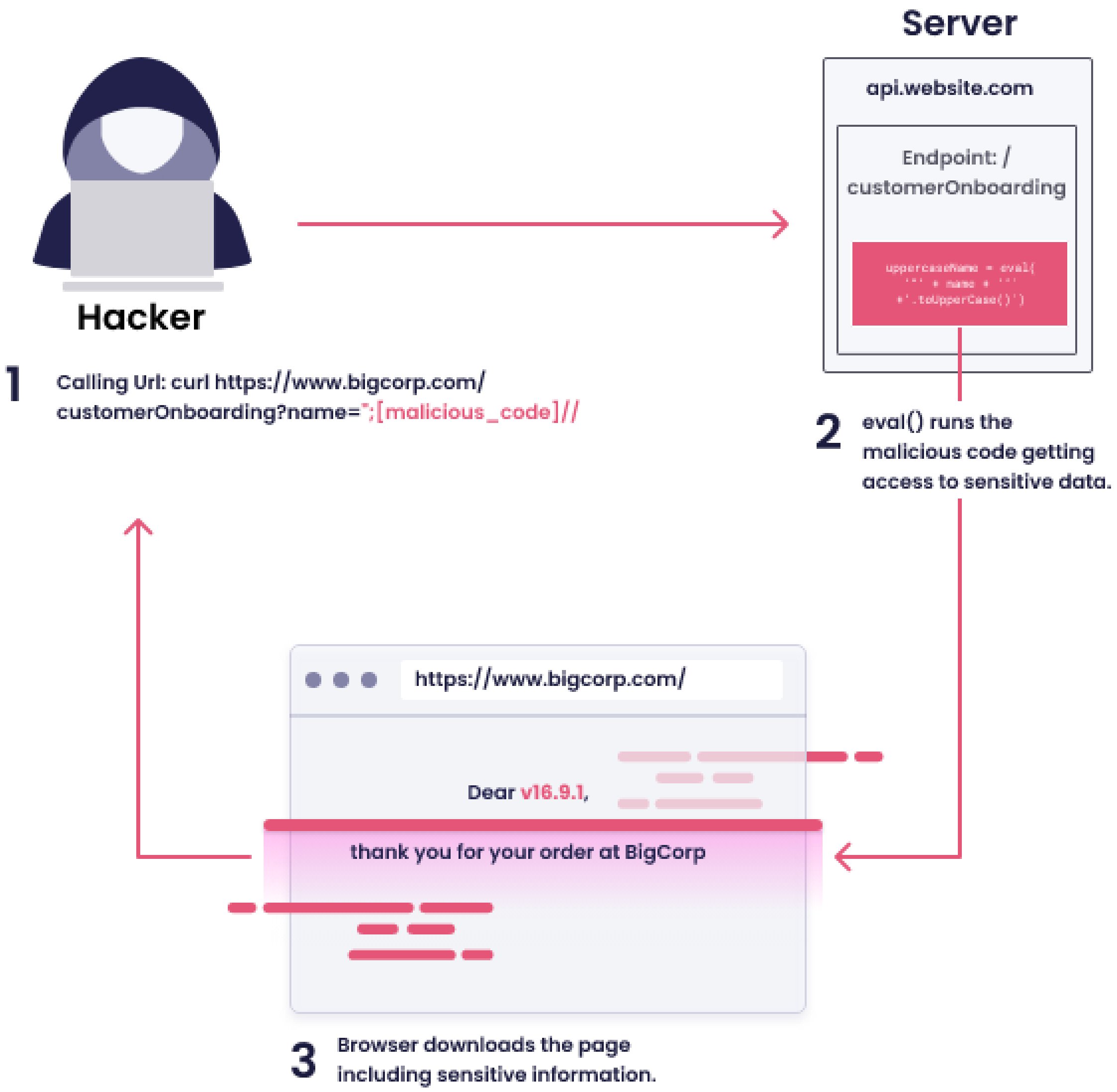


It is a flaw that arises when software that writes data to a buffer surpasses the buffer capacity, resulting in overwriting of neighboring memory locations. That is, too much information is transmitted to a repository that does not have enough space, and this information is gradually replaced by neighboring repository data.
For example, a buffer for login data can be configured to require an 8-byte username and password to be entered, so if a transaction contains 10 bytes (i.e., 2 bytes more than expected) input, the program can write down excess data over the buffer limit.

A.5 Code Vulnerability: Insecure Input

Code Injection





Example of Code Injection

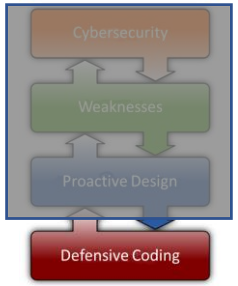
Code injection is a type of attack that allows an attacker to **inject** malicious code **into an application** through a **user input field**, which is then executed on the fly.

Code injection vulnerabilities are **rather rare**, but when they do pop up, it is often a case where the **developer** has attempted to **generate code dynamically**.

Preventing code injection attacks usually comes down to **reconsidering** the **need to dynamically execute code**, especially where user input is involved.

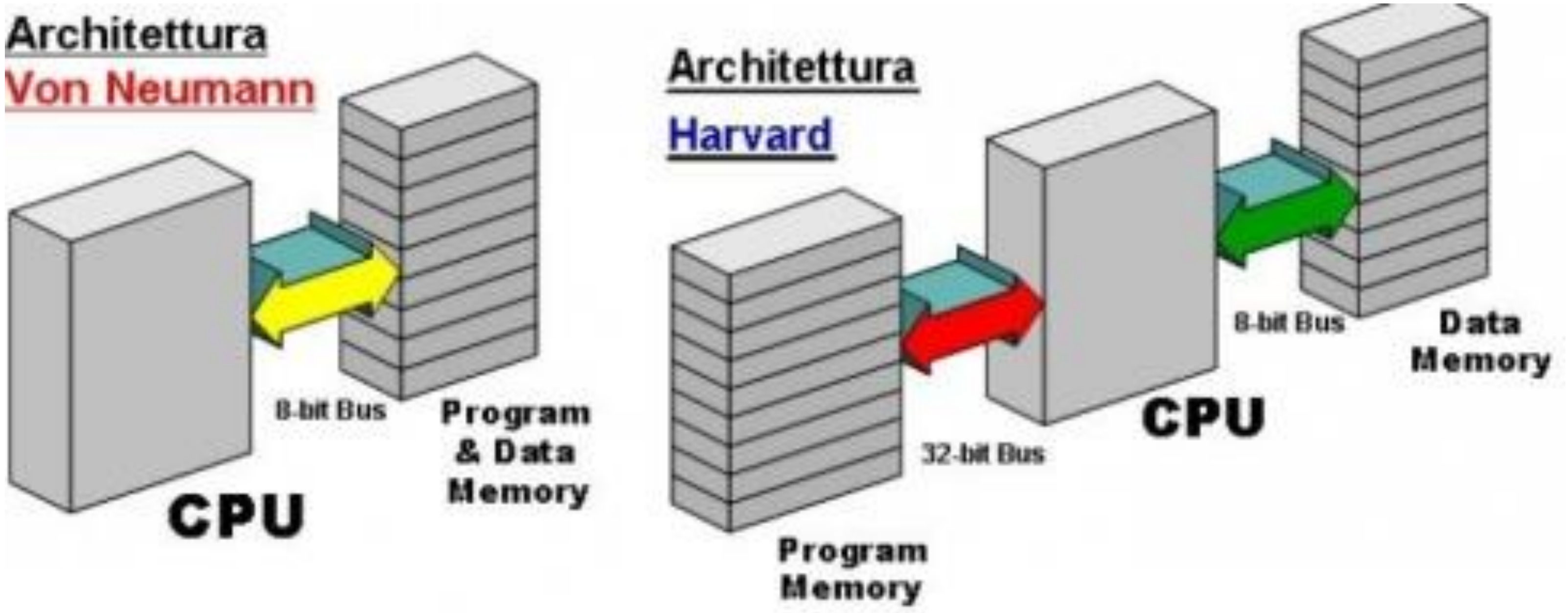
A.5 Code Vulnerability: Insecure Input

von Neumann vs Harward Architecture



Tricking an application to treat provided data as code

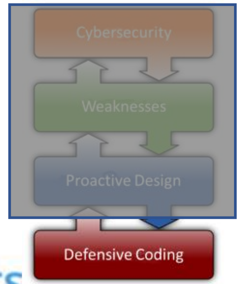
von Neumann vs Harward



Program & Data together → Metadata

Program in a place, Data in another → Limited interactions

A.5b Code Vulnerability: Buffer Overflow, Insecure Input



Secure Software Alliance

SSA Goals

•Creation of **software security awareness** at all levels in the organization

•Stimulate activities that contribute to **increase software security**.

•Trustee of the (open source) **Secure Software Framework**

•Develop a **secure software certificate model** for software based upon a positive advice from an inspection-organization accredited by the SSA.

•Follow and contribute to (international) **initiatives** in the area of **secure software development**

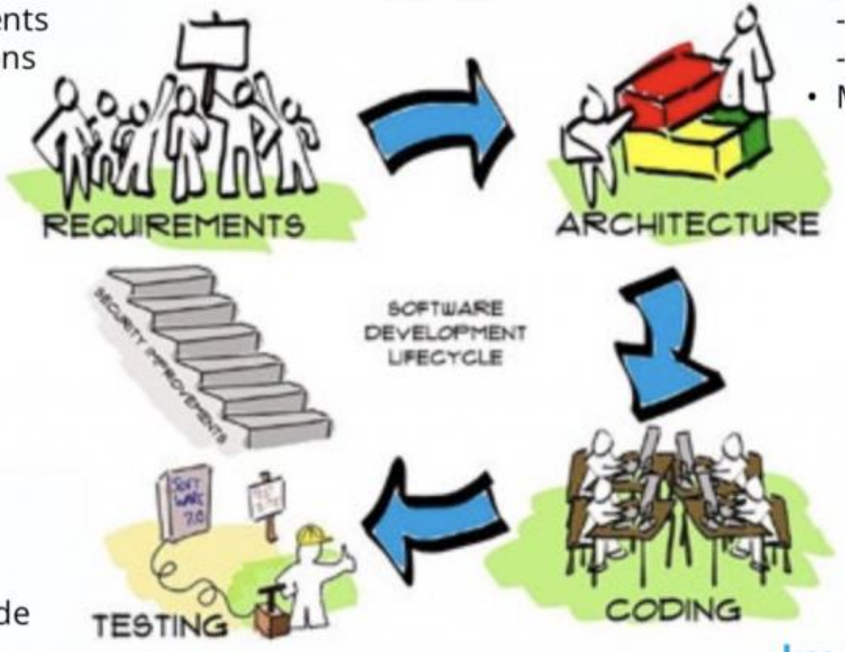
•Work **together with** other private and public **organizations** with similar interests.

Context

- Functions and environment
- Application assets
- Security requirements
- Security assumptions

Threats

- Functional threats
- Architectural threats
  - architecture inventory
  - threat library
- Mitigations



Verification

- Verification methode
  - code review
  - penetration test
  - vulnerability scan
  - fuzzing
  - abuse tests
- Verification process

Implementation

- Secure coding principles
- Secure coding standards
- Code audit