SECURITY – prof. Zanero

Jacopo Pio Gargano

Buffer Overflow

Def: reads an user supplied string of arbitrary length and copies it in a stack buffer.

Stack: high low addresses and growth, function vars, saved EIP, saved EBP, canary (implicit), struct

Changed stack: something, NOP SLED, shellcode, canary, AAAA, addr of nop sled (EIP)

Non-executable stack: ret to libc technique → put addr of system in EIP, AAAA, addr of ////bin/secret (that can be written in the stack)

Correct canary implementation: write the address of the saved EIP where you will then write the address of the NOP SLED

Env variables: use them when there is not enough space. Save a variable $EGG= NOP SLED + shellcode, then put the addr of the env variable in the saved EIP.

Special modes

- *W^X non executable stack*
  - Can't jump to shellcode b/c stack is non-executable
  - Can still overwrite saved EIP with address of system (ret to libc) and obtain a shell
- *ASLR address space layout randomization*
  - Can't do anything b/c we don't know the address of system and not even the address of the stack

Canary: a strategy to detect buffer overflow attacks at runtime that consists in adding a value (canary) to the stack frame between the return address and the local variables of the called function and checking that the value was not overwritten at function epilogue.

> *Correct implementation*: the value should be randomized and placed in a register
>
> W*rong but working implementation*: the value is hardcoded, and therefore it could be retrievable from the binary file / debugger.

Format String Vulnerability

Def: when there are variadic functions (the # of parameters is chosen at runtime) that can be set by the user and that allow them to read/write from/to memory. The attacker (untrusted user) will supply a formatted string so as to exploit the vulnerability.

Algorithm:

- What to write: 0x42344467 → split in half, convert each to decimal, pick the smallest for N1
- Where to write: 0x42414515
- Displacement = 7
- Compute the following:

    - <tgt> = where to write - <tgt+2> = where to write +2 - <pos> = displacement
    - <N1> = smallest_base10 - 8
    - <N2> = biggest_base10 - smallest_base10
- If 4234 is smaller than 4467 then <tgt+2> then <tgt>, else normal:
- <tgt> <tgt+2> % <N1> c% <pos> $hn% <N2> c% <pos+1> $hn

SQL Injection

Def: happens when there is a data flow from a user-controlled HTTP variable (parameter, cookie, header field) to a SQL query, without appropriate filtering and validation. The SQL query structure can be modified.

UNION → must have same # of columns and same column type as first query

JOIN → SELECT a.id, u.name FROM users u JOIN accounts a ON u.id = a.id WHERE a.name = 'Jon'

Remove vulnerability

- Use prepared statements instead of concatenating user input to a hardcoded SQL query.
- Use the DBMS/language provided escape functionality to escape input before composing the query (escape/blacklist SELECT, JOIN, UNION…)
- *CAN'T CHANGE THE CODE*: assign different privileges to tables, forbid the application level code to execute a SELECT.

XSS - Cross Site Scripting

Def: a vulnerability through which client side code can be injected in a page leading to: cookie theft, session hijack, manipulation of a session and execution of fraudulent transaction, snooping on private info, effectively bypass same origin policy, drive by download.

Stored or persistent: the attacker input is stored on the target server such as in a database, message form, visitor log, comment field. The victim retrieves the data stored from the web application without the data being made safe to render in the browser → the script is executed.

Reflected or non-persistent: user input is immediately returned by a web application in an error message, search result. The input is provided by the user as part of the request without that data being made safe to render in the browser and without permanently storing the user provided data.

E.G. request a username containing malicious script <script> alert(document.cookie)</script>

DOM based: the user provided data never even leaves the browser, so the attack payload is executed as a result of modifying the DOM environment in the victim's browser.

CSFR – Cross Site Request Forgery

Def: forces a user to execute unwanted actions (state changing actions) on a web application in which he is currently authenticated through a cookie/token. Malicious requests are sent to the vulnerable app through the victim's browser.

E.G. an attacker could setup a form that submits a request to send a message

Remove vulnerability

- Include a CSRF token with every request: session tokens that are associated to a user's session so they are unique. They are regenerated at each request that changes a state. For the Same Origin Policy, attackers are not able to read the secret CSRF protection token

CSP: The CSP is a header meant to limit the origin of the resources embedded in a webpage, such as scripts, fonts, images; it is mainly used as a mitigation against XSS as it can define some scripts or origins as trusted or not for serving scripts.

SOP: The SOP limits the allowed scripts to those contained in a first web page to access data/resources in a second web page only if both web pages have the same origin.

CORS: a mechanism that allows many resources on a page to be requested from another domain outside the domain from which the resource originated.

Network Protocol Attacks

DOS- denial of service – against availability → service unavailable

Sniffing – against confidentiality → get packets

Spoofing – against integrity and authenticity → pretend you are someone else

DOS

- Killer packets: Ping of death: very few or 1 packets can kill a service → pathological ICMP echo request that exploits a memory error in the protocol implementation → send a packet that exceeds the maximum legal length.
- Killer packets: Teardrop: exploit vulnerability in TCP reassembly → while reassembling kernel can hang/crash
- Killer packets: Land Attack: send a packet to itself (source is yourself) → loop and lock up TCP/IIP stack (Windows 95)
- Flooding: SYN Flood attacks: an attacker generates a lot of SYN requests which require each a SYN ACK → waste bandwidth to receive SYN and to send SYN ACK and also wasting memory for taking into account all of the half open connections. SYN COOKIES: the server is never keeping a list of half open connections, when it sends the SYN ACK then it stores a number to encrypt a key in the packet so that when it receives the ack of the syn ack the ack itself will tell it if it was a valid ack.
- Distributed DoS: many attackers flood the server → at a certain point in time everyone would send a request (navigate to a website) to the same target
    - Botnet: computer infected with malware and they all connect to a Command&Control server so that the attacker can send commands to the bots
    - Smurf: the attack has 2.1.1.1 as IP address, the target has 3.1.1.1 The attacker sends a ping with source 3.1.1.1 (target) to 1.1.1.255 (broadcast) → the router receives the request and the expected behavior is that every machine on the network replies to you (to the target)

SNIFFING

- Normally a Network Interface Card (NIC) intercepts and passes to the OS only the packets directed to that host's IP. In **promiscuous mode** the NIC passes to the OS any packet read off the wire.
- Mitigation: Use switched networks as opposed to hub-based networks. Switches selectively relay traffic to the wire corresponding to the correct NIC (ARP address based).

SPOOFING (ARP) [only works on LAN]

- ARP: Address → maps 32bit IP to 48bit MAC address
- First machine that answers to an ARP request is trusted
- The attacker sends an ARP reply to everyone (broadcast) saying that a certain IP is at his physical address → the first who replies so sooner or later the victim will trust you.
- CAM table → used by switch to know which MAC address on which port → attacker can generate a lot of packets and fill the CAM tables → cannot cache ARP replies and must forward everything to every port.
- Spanning Tree Protocol STP avoids loops on switched networks by building a Spanning Tree → switches decide how to build the ST by exchanging bridge protocol data unit packets to elect the root node → bridge protocol data unit packets are not authenticated, so an attacker can change the shape of the tree

for sniffing or ARP spoofing. (STP is useful to avoid loops → what if there is a loop among switches and a broadcast ping is sent?)

- The IP source address is not authenticated (you can send packets with any source you desire) → changing it in the UDP or ICMP packets is easy → the attacker won't see answers because they will be sent to the spoofed host (the machine the attacker impersonated) [blind spoofing] → if the attacker is on the same network tough, they can sniff the rest
    - TCP Blind Spoofing Attack: pretend you are source (ARP spoof), intercept a SYN ACK response by the victim, keep real source offline (DDOS), read the sequence numbers and then you can pretend being the source with no problem :)
- MITM: Man in the middle → all the attacks where an attacker can impersonate the server w.r.t. the client and vice-versa.
    - Physical or logical
    - Full or half-duplex (direction of the flow – blind MITM)
    -
- DNS Poisoning Attack: Domain Name Service is on UDP and the messages are not authenticated.
    - When a request to a website is performed:
        - If the answer is cached, the DNS answers
        - If there is no answer in cache: ask the DNS to recursively ask other DNS servers or (iteratively) the DNS will reply with the authoritative server for the domain you are trying to resolve
    - How to poison:
        - The attacker makes a recursive query to the victim DNS server.
        - The victim DNS contacts the authoritative server
        - The attacker impersonates the authoritative DNS server and spoofs the answer (the reply that it sends is what he wants)
        - The victim's DNS is poisoned → holds a malicious record
        - The victim is redirected to the malicious website
    - However: in the spoofed answer we need to use the ID of the DNS transaction initiated by the victim.
- DHCP Poisoning: DHCP is not authenticated. With a single spoofed DHCP response, the attacker can set IP address, DNS addresses, default gateway of the victim.
- ICMP Redirect: in theory it tells a host that there is a better route for a given destination → the attacker can forge an ICMP redirect packet to elect his computer as the gateway → intercept a packet in the original connection → create a half-duplex MITM
- Route Mangling: if the attacker can announce routes to a router, they can do many things:

CONCLUSIONS:

- Certain DoS attacks exploit memory errors in the network stack implementations.
- Dos is generally always feasible, given enough resources
- Attack are possible essentially by the lack of authentication in the protocols

Attacks

Firewall

- Attacks:
  - DNS: check if firewall whitelists address (then yes) or IP (then no)
  - GMAIL or anything over HTTPS: can't decrypt or get a valid certificate
  - SOLUTION TO Malware download over HTTP: enforce all traffic to pass via a proxy and scan the payload with an AV + AV on endpoint. Software house can deliver the update through HTTPS + digital signature
  - ARP
    - PROBLEM: the protocol lacks authentication
    - SOLUTION TO ARP spoofing: the switches can block any ARP response with the gateway's IP address and from a MAC address different than the gateway's own address.
    - DETECT ARP: detect duplicate ARP responses with different MAC addresses at network switches
  - ISN Initial Sequence Number MUST be random, otherwise can be predicted
  - DHCP
    - SOLUTION: tag each port of a switch with client or server and allow only servers to send DHCP offer packets

Amplification-based DoS

- Calculate the amplification factor (response length / request length)
- IDEA: attacker on network A spoof its IP address with the victim's one, and sends many requests to a server B running this protocol. B will reply to the spoofed IP (i.e., to the victim) with 256 times the data sent to B by the attacker, possibly saturating the victim's bandwidth.
- The attacker just needs a bandwidth that is >= than the victim's bandwidth /n.
- SOLUTION: switch to TCP that is immune to the amplification issue due to three way handshake: the victim will receive a SYN-ACK but won't answer because it didn't initiate any connection.

Secure Network Architectures

Firewall

- network access control system that verifies all the packets flowing through it, must be the single enforcement point between a screened network and outside networks. Its functions are:
    - IP Packet filtering
    - Network address translation NAT
- Powerless against insider attacks (use the machines that are connected to the internet, e.g. 3G, and are also connected to the LAN)
- Firewalls can be violated since they are computers
- Firewalls are guided by policies/rules:
    - Policies are built on a default deny base: "deny everything except"
- Layers:
    - Stateless Packet filters (network layer)
        - Decodes the IP and part of the TCP header (source and destination IP and port, protocol type, IP options) but cannot track TCP connections between different packets, no payload inspection
        - Every network packet filter allows to express the concept of "if packet matches certain condition then doth the following (block/allow/log/other..)
        - If you have a deny all policy and then an allow only on a certain port/IP then it's pretty useless since you can't talk with the outside but just receive
    - Stateful packet filters (network-transport layer)
        - Include packet filter + **keep track of the TCP session**
        - We can track connections without adding a response rule
        - Performance bounded on a per-connection, not on a per-packet basis
        - Application-layer filtering is possible, for example can disallow specific objects from being sent/received.
        - UDP → a session can be inferred for NAT (if response packet received within a certain timeout it is considered "in the session")
        - Some protocols transmit network information data at application layer (which is the wrong layer) → stateful firewalls must take care of these.
        - Firewalls can inspect data and detect intrusions.
    - Circuit level firewalls (transport-application layer)
        - Relay TCP connections → client connects to a specific TCP port on the firewall, which then connects to the address and port of the desired server.
        - Essentially it is a TCP-level proxy. It is the same as a stateful packet filter, but transparently.
        - ONLY HISTORICAL EXAMPLE → SOCKS
    - Application proxies (application layer)
        - A proxy that acts the same as circuit firewalls but at application layer.
        - Benefits
            - inspect, validate, manipulate protocol application data.
            - Can authenticate users, apply specific filtering policies, perform advanced logging, content filtering or scanning (anti-virus/spam)
        - Usually used to improve performance

Architectures for secure networks

- Dual-zone architectures
    - Castle model: whatever is inside is good, whatever is outside is bad, there are walls to protect the inside.
        - BUT there are some resources that must be accessible from outside (services we offer)
        - BUT there may be the need of remote users to access the corporate network
    - PROBLEM: if we mix externally accessible servers with internal clients, we lower the security of the internal network.
    - SOLUTION: we allow external access to the accessible servers, but not to the internal network → split the network by privilege-levels → HOW? → firewalls
        - Create a semi-public zone called DMZ (demilitarized zone)
        - This zone will host public servers (web, FTP, public DNS server, intake SMTP)
        - This zone won't host critical or irreplaceable data and services
- VPN: Virtual Private Network
    - Requirements:
        - Remote employees need to work as if they were in the office, accessing resources on the private zone → Road warrior connection
        - Connecting remote sites without sites without using dedicating lines → Site2Site connection
    - Concept: encrypted overlay connection over a public network
    - Full tunneling
        - Every packet goes through the tunnel
        - Waste of bandwidth: every time the request goes first to the VPN, then to the server even for things that are not needed (e.g. Netflix).
        - Good from a security point of view
    - Split tunneling
        - Network traffic directed to corporate network directed to VPN
        - Network traffic to the Internet is directed to ISP (internet service provider)
        - More efficient, less control

Network Security – SSL, TLS, SET

Issues of Transaction Security

- Problems of remoteness:
  - Trust factor between parties
  - Use of sensitive data
  - Atomicity of transaction
- Internet protocol problems
  - Authentication
  - Confidentiality
  - Transparence and critical mass problem

SSL Secure Sockets Layer

- Originally designed by Netscape
- Enforces:
  - Confidentiality and integrity of the communications
  - Server authentication
  - Client authentication (almost never done)
- Uses both symmetric and asymmetric crypto for performance reasons
- HANDSHAKE PHASES:
  1. CLIENT→SERVER Cipher Suite + random data:
     a. Client sends to server its list of known suites of algorithms in order of preference → Algorithms: key exchange, bulk encryption, message authentication code, pseudorandom function
  2. SERVER→CLIENT Cipher selection + other random data + server certificate
     a. Validation of server certificate:
        i. Certificate validity period
        ii. Root CA trusted?
        iii. Certificate valid or revoked?
        iv. Is the name of the server in the certificate the same the client requested?
  3. CLIENT→SERVER Pre-master secret (encrypted with server public key) + [client certificate signed with client private key → the server checks the certificate in the same way]
     a. SERVER and CLIENT **compute the shared key** from:
        i. pre-master secret
        ii. client random data
        iii. server random data
     b. NOTE: random data is used to make sure that every time this exchange is played it will be different → resistant to replays
- MITM attack
  - Can see the traffic → can read the client and server random data, but does not have the shared key as it lacks the pre-master
  - Not possible because MITM could send a fake certificate but it would not be trusted by CA authority, or could substitute the public key which would still make the certificate invalid
- PROS:
  - Protects transmissions → confidentiality and integrity of data

    - Ensures authentication of server and optionally of client
- CONS:
    - No protection before or after transmission
    - Relies on PKI Public Key Infrastructure
    - Not foolproof → if users authorizes browser to go ahead when receive a warning on certificates…..

SET Secure Electronic Transaction

- Created by VISA+MasterCard
- Protects transactions, not connections
- CONCEPT
    - Cardholder sensa the merchant only the order of goods (amount to pay)
    - Sends the payment data to the payment gateway
    - Empower gateway to verify correspondence
- Dual signature (computed by gateway)
    - Order information → HASH1
    - Payment instruction → HASH2
    - DUALHASH = Hash (HASH1,HASH2)
    - {DUALHASH}CardholderPrivate → Dual signature
- Gateway → Merchant: {order info + HASH2 + {DUALHASH}CardholderPrivate }MerchantPublic
    - Merchant has HASH1 (compute from order info) and HASH2
    - Merchant can hash HASH1 and HASH2 obtaining DUALHASH and can check it is the same as {DUALHASH}CardholderPrivate by decrypting it with CardholderPublic
    - Merchant needs this to perform a check
- Gateway → Merchant: {payment instructions + HASH1 + Dual Signature}GatewayPublic
- Merchant → Bank: {payment instructions + HASH1 + Dual Signature}GatewayPublic
    - Bank has HASH2 (compute from payment instructions) and HASH1
    - Bank can hash HASH1 and HASH2 obtaining DUALHASH and check it is the same as {DUALHASH}CardholderPrivate by decrypting it with CardholderPublic
    - Bank needs this to perform a check and authorize the payment
- PROBLEM: distribution of keys to cardholders → does not scale

Malware

Malware = Malicious Software

- Definition: code that is written to violate a security policy
- Categories:
  - Virus → self-propagate by infecting other files, usually inside other executables
  - Worm → program (i.e. an executable) that self-propagates, exploits host vulnerabilities
  - Trojan horses → apparently benign program that hide a malicious functionality and allow remote control
- Lifecycle:
  - Reproduce and Infect
    - find balance between infection and possibility of being detected
    - find a suitable propagation vector (social engineering, vulnerability exploits)
  - Stay hidden → the longer it stays hidden the more likely it is to infect others and propagate
  - Run payload → sometimes harmful
- Techniques:
    - Boot viruses
      - When computer boots, it scans all the drives and looks at the first sector of each drive → there can be a Master Boot Record MBR → the MBR is executed
    - File Infectors
      - Simple overwrite virus → viruses that infect another exe
      - Parasitic virus → append code and modify program entry point
      - Multicavity virus → inject code in unused regions of program code
- Macro viruses
  - Macro functionalities blurs the line between data and code
    - For example spreadsheet macros can modify a spreadsheet, modify other spreadsheets, access address book, send email…
- Mass-mailers → rebirth of worms
  - Email software started allowing attached files including executables, masqueraded executables (love-letter-for-you.txt.vbs → the 'vbs' extension is not shown by windows)
  - Spread by emailing itself to others when executed
  - Social networks "have you seen this video of yourself?"
- Mass Scanners → modern worms
  - Infect a computer and seek out new targets (since you don't have a list of who's connected anymore) → randomly try IP addresses
- DEFENSES AGAINST WORMS:
  - Patches
    - Most worms exploit known vulnerabilities
    - Useless against zero-days worms
  - Signatures → blacklisting
    - Problem: malware propagates very fast → no time to develop a signature
  - Intrusion or anomaly detection
    - Notice fast spreading, suspicious activity → if I notice a program that I do not recognize and looks like a malware
- Antivirus and anti-malware
  - Basic strategy: signature-based detection

        - Database of byte-level or instruction-level signatures that match malware (use of wildcards, common regexs) → does not really work anymore → earlier malware consisted only of viruses which are pieces of code that duplicate → easy to put them in a database and remove them → change to non self-replicating malware → different on every machine
    - Heuristics (check for signs of infection):
        - Code execution starts in last section
        - Incorrect header size in portable executable header
        - Suspicious code section name
        - Patched import address table
        - PROBLEM: only works with viruses
    - Behavioral Detection
        - Finding a difference in the way malware behaves w.r.t. normal programs
        - Find common malware behavior
- Virus stealth techniques → malware needs to be not easy to find
    - Entry Point Obfuscation
        - Virus hijacks control after the program is launched, overwriting libraries and functions
    - Polymorphism
        - Change the payload at each infection
        - Encrypt payload using a different key for each infection
        - Antivirus can detect encryption routine
    - Metamorphism
        - Create different versions of the code that look different but do the same semantics
    - Dormant period
        - No activity is exhibited during this period
    - Event-triggered payload
    - Encrypting/Packing
        - Encrypt malicious content changing key at each execution
        - Anti-debugging and anti-virtual machines techniques → evasive malware (avoids analysis, not detection)
    - Anti-virtualization techniques
        - If a program is not run natively on a machine, probably…
            - …it is being analyzed in a security lab
            - …it is being scanned inside a sandbox of an antivirus
            - …it is being debugged by a security specialist
        - Detecting Virtual Machine / hardware supported VM → easy
        - Emulator → theoretically undetectable, practically easy to detect
- COUNTERACTING MALWARE
    - Suspicious executable is reported by someone → automatically and manually analyzed → antivirus signature developed
    - Analysis
        - Static analysis
            - Parse the executable code
            - PRO: More code coverage, analysis of dormant code
            - CON: More obfuscation leading to metamorphism, encryption, packing
        - Dynamic analysis

        - Observe the runtime behavior of the exe
        - PRO: Less obfuscation
        - CON: less code coverage, less analysis of dormant code
- Rootkits → disappearing
    - Historically: you become a root on a machine and you plant your kit to remain root
        - Make files, processes, user and directories disappear
        - Make the attacker invisible
    - Can be either userland or kernel-space
        - Backdoored login
        - Trojanize to hide processes in Task Manager
    - Userland → easier to build, but often incomplete, easier to detect
    - Kernel space → more difficult to build, but can hide artifacts completely
    - How to detect rootkits
        - Only if machine exhibits weird behavior
        - Post-mortem on different system
        - Trusted computing base, tripwire
        - Cross-layer examination

Mobile Security

- Smartphone are considered as a good target
  - Always online, always computing resources available, handle sensitive data (emails, social network, banking info, location)
- iOS
  - code signing (rooted in Apple signing key) → means you can't use your own (attacker's) code
  - sandboxing with predefined permission
  - closed ecosystem (App Store is CA)
- Android
  - code signing checked only at installation of apps, not after when it could be modified
  - sandboxing with explicit permissions
  - open ecosystem (no PKI Public Key Infrastructure, no CA)
- Sandboxing → applications are not able to interact with each other unless authorized by the kernel
  - iOS → the kernel uses MAC (mandatory access control) to restrict the files and memory space that each app can access → every app can access only its files and memory
  - Android → each app is assigned a distinct user and Linux takes care of the privilege separation automatically.
- Permission-based Authorization
  - iOS → use privacy profiles to define access control rules (e.g. Safari can access Address Book). User must confirm at first use. Default deny policy
  - Android → each app requests explicit permissions that the user must approve BEFORE installation. Approve all or deny all.
- Code Signing
  - iOS → the kernel enforces that only signed code is executed (app is signed with developer's key, developer's certificate is signed by Apple's CA, no external code allowed), no dynamically generated code, no traditional shellcode, sandbox ensures last line of protection.
  - Android the installer checks that every new app is cryptographically valid. Apps are verified only upon installation, not each time they are run (if updated maybe code changes). Apps can be signed with self-signed certificate since there is no CA or PKI. Dynamically generated code is allowed → silent updates. Sandbox ensures last line of protection.
- Attacks
  - Call or text to premium numbers → $
  - Silently visit web pages → $
  - Steal sensitive info → $
  - Turn the device into a bot → $
  - Lock device and ask for a ransom → $
- Mitigate Mobile Malware
  - Google play runs automated checks
  - SMS/call blacklisting and max quota
  - Google App Verify
  - Sandboxing
  - Anti-malware are pretty useless because the antivirus signature is developed after the app is removed from the official store