1. TCP/IP

7 Layer:

- Physical Layer (1): Mit Channel verbinden; Bytes übertragen; z.B. Repeater, Hub
- Data Link Layer (2): Fehlerkontrolle zwischen angrenzenden Knoten; z.B. Bridge, Switch (Ethernet)
- Network Layer (3): Übertragung und Routing zwischen Subnetzen; z.B. Router (IP)
- Transport Layer (4): Ordnen, Multiplexing, Korrektheit; (TCP/UDP)

Physical Layer (1)

Hardware (Twisted Pair, Coax, Wireless transceiver, optisches Kabel…); schwierig zu kontrollieren

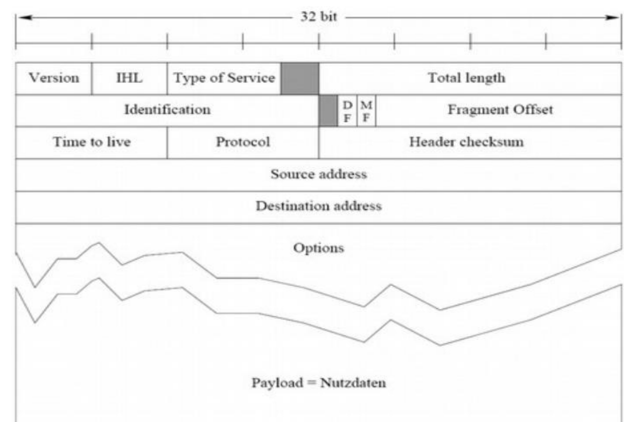
Data Link Layer (2)

dest (6 Bytes)	src (6 Bytes)	type (2 B)	data (46-1500 B)	CRC (4 B)

Adressen sind 48 Bits lang (z.B. 00:38:AF:23:34:0F) und durch den Hersteller hartcodiert (=MAC-Adresse)
type (2 Bytes) gibt an welches Protokoll gekapselt wird (IP, ARP, RARP…); CRC = Prüfwert
Tools: Wireshark um Netzwerkverkehr zu erfassen; ipconfig um MAC-Adresse anzuzeigen und zu ändern

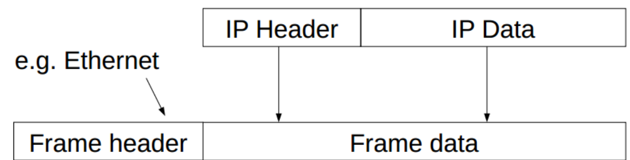
Network Layer(3)

Das Internetprotokoll (IP) ist das Bindeglied zwischen Hosts im Internet. Es ist verbindungslos (connectionless) und die Ankunft, Integrität, Ordnung und Nichtduplikatur (non-duplication) werden nicht garantiert ➔ Pakete können, zumindest in IPv4, verloren gehen, nochmals gesendet oder verändert werden



Der IP-Header besteht aus 20 Bytes, davon
4 Bits für die Version
4 Bits für die Headerlänge (IHL) (gibt an wie viele 32-Bit-Wörter im Header sind (inklusive IP options)),
7 Bits für den Type of Service (davon 3 Bit für Priority und 4 für QOS)
16 Bits für die Total length (gibt Länge des gesamten Pakets an (inkl. Kopfdaten))
16 Bits für die Identifikation (für die Reassembly verschiedener Pakete) (werden +1 inkrementiert)
3 Bits für Flags: Bit0: reserviert (muss 0 sein); Bit1: (Don't Fragment: wenn 1 ➔ Paket darf nicht fragmentiert werden); Bit2: (More Fragments: wenn 1 ➔ es folgen noch weitere Fragmente)
13 Bits für den Fragmentoffset (gibt an ab welcher Position innerhalb des Pakets das Fragment anfängt)
8 Bits für Time to live (jede Station verringert Wert um 1; ist 0 erreicht, wird Paket verworfen)
8 Bits für das Protokoll (gibt an welchem Folgeprotokoll die transportierten Nutzdaten angehören (TCP=6, UDP=17))
16 Bits für die Prüfsumme des Kopfdatenbereichs (wird bei jeder Station verifiziert)

Direct IP delivery



Host sind in lokalem Netzwerk direkt miteinander verbunden. Das Problem hierbei ist, dass der Link Layer 48 Bit Ethernetadressen verwendet und der Network Layer 32 Bit IP-Adressen. Um trotzdem ein IP-Paket schicken zu können, wird dieses in ein Ethernetpaket eingekapselt (dabei muss die IP-Zieladresse auf die Ethernetadresse gemapt werden.

Address Resolution Protocol (ARP)

Mapt Netzwerkadressen (3) auf Link-Level-Adressen (2). Wenn Host A die assoziierte Hardwareadresse (MAC) einer IP-Adresse von Host B haben möchte, schickt er eine Broadcast ARP-Nachricht über den Physical Link Layer (1). In dieser Nachricht gibt A sein eigenes Mapping mit an. B antwortet A dann mit einer ARP-Antwortnachricht.

Tools: ipconfig/ifconfig/ip/arp/ping

ARP Message Format

hardware type (2 byte)		protocol type (2 byte)
hw.adr.size (1 byte)	prot. adr. size (1 byte)	opcode (2 byte)
sender Ethernet address (6 byte)		
sender IP address (4 byte)		
target Ethernet address (6 byte)		
target IP address (4 byte)		

Fragmentation

Wird verwendet wenn die Kapselung von Protokollen zu niedriegerem Level eine Aufsplittung erfordert (z.B. wenn Datagrammgröße größer als Data Link Layer MTU ist). Jedes Fragment wird als eigenes IP-Datagramm übertragen (mit 2 Flags und 13 Bit Offset). Wenn ein Fragment fehlt oder falsch ankommt, wird das gesamte Datagramm verworfen.

Attacken auf Layer 2 und 3

Fragmentierungsattacken

Ping of death (Teardrop attack)

Die maximale Größe eines IP-Datagramms wird missachtet. Pings nutzen normalerweise 64 Bytes. Mittels Fragmentierung kann stattdessen ein IP-Paket mit einer Größe > 65535 Bytes gesendet werden. Dadurch kommt es bei der Zusammensetzung der Pakete auf Empfängerseite zu einem Overflow.

IP fragment overwrite (fool the firewall)

IP-Datagramm das Layer 4 Daten kapselt (z.B. TCP) wird fragmentiert. Layer 4 Header beinhaltet erlaubten Port (z.B. 80) und wird deswegen von der Firewall durchgelassen. Die Daten werden fragmentiert gesendet. Das nächste Paket hat den frag-offset auf 1 gestellt. Dadurch werden Header (inklusive Port) überschrieben (z.B. auf neuen Port=23). Nachdem die Pakete wieder komplett zusammengestellt wurden, werden sie über den neuen Port ausgeliefert.

Gegenmaßnahmen

Die Pakete auf der Firewall wieder zusammenführen; Sanity-Checks auf den IP-Headern durchführen; Betriebssystembugs fixen

LAN Attacken

LAN-Attacken versuchen Informationen auszulesen, sich als einen falschen Host auszugeben oder Zustellmechanismen zu manipulieren. Dies wird duch Sniffing, IP-Spoofing und ARP-Attacken realisiert.

Netzwerksniffing

Angreifer lauscht bei Datentraffic mit. Viele Protokolle übertragen die Authentifizierungs-informationen im Klartext ➔ Usernamen und Passwörter können gesammelt werden. Ist vor allem bei kabellosen Netzwerken problematisch.
Sniffing ist auch bei switched Ethernetverbindungen möglich, wo der Switch Pakete nur an richtigen Empfänger weiterleitet:

MAC Flooding: Switch hat interne Tabelle für MAC-Adressen und Port Mapping. Wird der Switch mit falschen MAC-Adressen überflutet, kann ein Overflow auf die Tabelle passieren. Manche Switchs gehen daraufhin in einen Hubmodus über.

MAC duplication/cloning: Man kann die MAC-Adresse des eigenen NICs neu konfigurieren, woraufhin der Switch diese neue Adresse speichert und den Traffic dorthin weiterleitet.

Tools: Wireshark (Sniffing, Decodiert Header, stellt fragmentierte IP-Pakete zusammen), macof (überflutet Netzwerk mit zufälligen ARP-Nachrichten), packeth (GUI um willkürliche Pakete zu erstellen)

Gegenmaßnahmen:

Sniffers: DNS-Test: Manche Sniffer versuchen Namen die mit IP-Adressen verknüft sind zu finden ➔ Traffic für falsche IP-Adresse generiern und schauen ob DNS-lookup für IP-Adresse stattfindet

MAC flooding: Port-Security verwenden (Anzahl der MAC-Adressen die sich pro Port am Switch verbinden dürfen limitieren); 802.1X (Packetfilterungsregeln werden durch einen zentralen Server ausgeführt); MAC-Filterung (Limiterung der Anzahl verschiedener MAC-Adressen)

ARP Poisoning

ARP verwendet keinerlei Authentifizierungsmethoden ➔ Race Conditions mit falschem Mapping möglich. Es können auch falsche ARP-Queries verschickt werden, wodurch falsche Informationen im Cache des Hosts gespeichert werden.

Mögliche Verwendungen: MITM-Attacke; DoS-Attacke (Ziel-IP auf nicht existierende MAC-Adresse mappen); Kann auch Gateway als Ziel haben (Gateway imitieren um gesamten Traffic zu filtern, oder Gateway IP zu nicht existierender MAC-Adresse mappen um gesamten ausgehenden Traffic zu blockieren)

Tools: ettercap/bettercap (ARP-Poisoning); packet (poisonous ARP-Pakete erstellen);
scapy (Paketmanipulationsprogramm)

Gegenmaßnahmen:

Statische ARP-Tabellen in LAN verwenden (oder zumindest für sensible Hosts); ARP-Antworten ignorieren die nicht angefragt wurden; Paketzustellung verbieten wenn MAC-Adresse mehreren Ports zugeordnet ist (dadurch ist jedoch DoS-möglich); Layer2-Verschlüsselung für kabellose Netzwerke (WPA2); physische Sicherheit für kabelgebundene Netzwerke

type (1 byte)	code (1 byte)	header checksum (2 bytes)
data		

Attacken auf Layer 3 (Network Layer)

ICMP (Internet Control Message Protocol)

Das ICMP wird für Kontroll- und Fehlernachrichten über die Zustellung von IP-Datagrammen verwendet. ICMP-Nachrichten sind in IP-Datagrammen eingekapselt und können Requests, Responses oder Error-msg sein.

ICMP Echo Attacken

Information gathering: ICMP echo Datagramme werden an alle Hosts eines Subnetzes gesendet. Angreifer sammelt Antworten und weiß dadurch welche Hosts aktiv sind.

Packet amplification (SMURF attack): sende manipulierten ICMP Echorequest (mit IP-Adresse des Opfers) ins Subnetz. Dadurch bekommt Opfer ICMP Echo Replies von jeder Maschine im Netz

Gegenmaßnahmen:

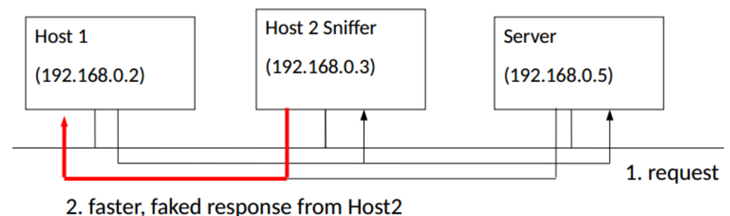
Smurf-Attacken sollten in echten Netzwerken nicht funktionieren (außer im LAN); Gateway übermittelt Broadcastpakete nicht; die Broadcastdomain endet beim Router
Firewallkonfiguration: erlaube ausgehende Requests und eingehende Replies

ICMP Destination Unreachable

Eine ICMP-Nachricht die von Gateways verwendet wird um zu vermitteln, dass das Datagramm nicht ausgeliefert werden kann (hat viele Untertypen (Network, Host, Protocoll, Port, ... unreachable)). Kann verwendet werden um (DoS) Knoten aus dem Netzwerk zu „schneiden" indem permanent falsche destination unreachable Nachrichten gesendet werden
Firewalling: Wenn ein Port geschlossen ist, sendet das BS normalerweise eine port unreachable ICMP-Nachricht. Firewalls tun dies oft nicht → wird ein firewalled port angepingt kommt es zu einem timeout, während geschlossene Ports eine ICMP-Nachricht produzieren.

IP Spoofing

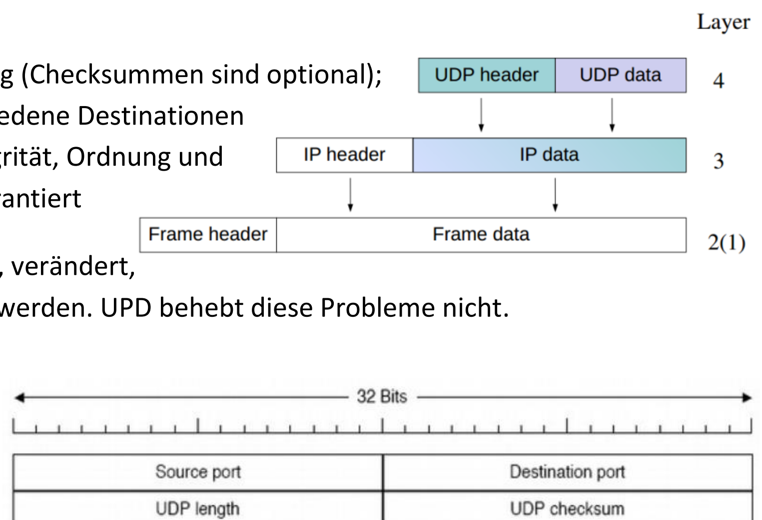
Es wird ein Datagramm mit einer falschen IP-Adresse als Absender geschickt (IP-Adressen werden nicht authentifiziert). Wird verwendet um (falsche) sicherheitskritische Informationen zu senden.



Attacken auf Layer 4 (Transport Layer)

UDP (User Datagram Protocol)

Baut auf IP auf; ist connectionless und unzuverlässig (Checksummen sind optional); implementiert port Abstraktion (es können verschiedene Destinationen mit selber IP adressiert werden); die Ankunft, Integrität, Ordnung und Nichtduplikatur (non-duplication) werden nicht garantiert



Durch IP-Netzwerke können Pakete fallen gelassen, verändert, in anderer Reihenfolge ankommen oder dupliziert werden. UPD behebt diese Probleme nicht.

UDP Spoofing

Genau wie bei IP Spoofing; Es wird die Absenderadresse des Opfers eingetragen wodurch eine Antwort zum Opfer gesendet wird

UDP Hijacking

Angreifer sieht, dass Opfer Anfrage an Server geschickt hat, und versucht schneller als der Server darauf zu antworten (mit Server IP-Adresse als Absendeadresse).

UDP Storm

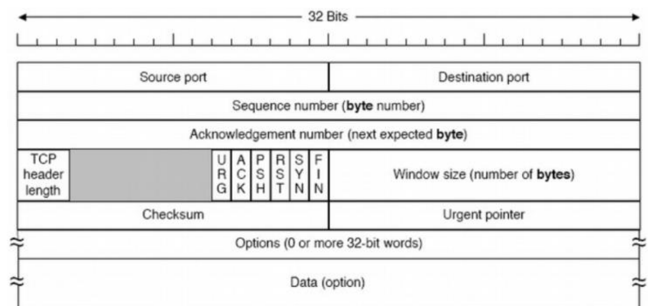
Wenn 2 Hosts einen replying UDP Service anbieten (z.B. echo, daytime, Qotd,...), kann ein UDP Paket so manipuliert werden, dass die Quell-IP die von Opfer B und der Quellport der vom Serviceports von Opfer B ist und die Ziel-IP die von Opfer A und der Zielport der vom Serviceport von Opfer A ist. Dadurch antworten die beiden Opfer sich immer gegenseitig.

UDP Portscan

Gehört zur Information-Gathering-Phase einer Netzwerkattacke. Es wird ein (zero-length) UDP-Packet an jeden Port eines Hosts geschickt. Kommt eine ICMP-error-Nachricht zurück, kann angenommen werden, dass der Service dieses Ports nicht verfügbar ist, sonst schon.
Viele TCP/IP-Stacks limitieren die Anzahl der maximalen Fehlernachrichten → der Scan kann sehr lange dauern (z.B. Linux: max. 80 msg alle 4 Sekunden)

TCP (Transmission Control Protocol)

Baut auf IP auf; ist connection-oriented, zuverlässig und implementiert die Port abstraction (<IP, Port> == Socket). Die Ankunft, Integrität, Ordnung und Nichtduplikatur (non-duplication) werden garantiert.



Sequence Number (seq): gibt die Position der Segmentdaten innerhalb des Kommunikationsstreams an (seq=1234 bedeutet, dass der Inhalt dieses Segments Daten ab der Position 1234 enthält)

Acknowledgement number (ack): gibt die Position des nächsten erwarteten Bytes an (ack=12345 bedeutet, dass die Bytes bis 12344 korrekt angekommen sind und als nächstes das Byte 12345 erwartet wird)

TCP Window: für flow control verwendet. Ein Segment wird nur akzeptiert, wenn seine Sequenznummer zwischen der letzten ack-Nummer und der letzten ack-Nummer + die window Size ist. Die Windowsize ändert sich dynamisch. Sie wird vom Empfänger gesetzt um zu sagen wie viel er empfangen kann.

TCP Flags: SYN: Anforderung zur Synchronisation (beim Aufbau der Verbindung genutzt)
ACK: Die acknowledgement number ist gültig; alle Segmente, bis auf das erste, haben dieses Flag gesetzt
FIN: Anforderung die Verbindung zu schließen
RST: Anforderung die Verbindung sofort zu reseten
URG: gibt an, dass der Urgent Pointer gültig ist
PSH: Anforderung eine Push-Operation auf dem Stream durchzuführen

TCP-Setup:
Server lauscht an einem Port; Client sendet Anfordeung an Server mit SYN-Flag und zufälliger sequence number c; Server antwortet mit Segment mit aktiven SYN und ACK- Flags, einer zufälligen

sequence number s und c+1 als acknowledge number; Client sendet Segment mit ACK-Flag und mit c+1 als sequence number und s+1 als acknowledge numer

Die Initial Sequence Number sollte wirklich zufällig und nicht vorraussagbar sein.

Delayed ACK: Wenn Daten empfangen wurden, wartet der Empfänger bis zu 200ms in der Hoffnung, dass noch mehr Daten kommen, bevor er ein ACK zurücksendet; Empfängt der Sender kein ACK (timeout), sendet er die Daten nochmal

Shutdown: Beide Partner können die Terminierung der Verbindung einleiten indem sie ein Segment mit dem FIN-Flag senden. Der andere Partner antwortet dann mit gesetztem ACK-Flag und danach wird der Partner, der die Terminierung eingeleitet hat, keine Daten mehr schicken (außer ACKs mit leeren Segmenten); Das nennt sich eine halb geschlossene Verbindung

TCP Scanning

In Information gathering Phase; checkt welche Ports eines Hosts offen sind. In der einfachsten Form wird einfach eine TCP-Verbindung zu dem Port geöffnet. Gelingt dies, wird davon ausgegangen, dass der Service verfügbar ist. (Im Unterschied zu UDP, werden keine ICMP-Pakete verschickt, wenn ein Port nicht erreichbar ist)

TCP SYN Scan (aka half open scanning)

Angreifer sendet SYN-Packet (kommt SYN/ACK zurück ist Port offen, kommt hingegen RST zurück ist der Port geschlossen). Als nächstes sendet der Angreifer ein RST-Paket anstatt eines ACK-Pakets zurück ➔ die Verbindung ist nie komplett offen und das Event wird vom BS nicht geloggt.

TCP FIN Scan

Angreifer sendet FIN-Paket. Die meisten TCP/IP Implementierungen (nicht Windows) antworten mit RST wenn der Port geschlossen ist und garnicht wenn er offen ist.

OS Fingerprinting

Um herauszufinden welches BS ein Host verwendet kann versucht werden untypische Pakete an ihn zu verschicken (z.B. verwenden von reservierten Flags im TCP-Header, Kombination verschiedener Flags im TCP-Header die keinen Sinn ergeben, ...). Nicht alle BS reagieren auf solche Anfragen gleich ➔ es kann auf das benutzte BS geschlossen werden.

Tools: NMAP (IP-Scans, UDP-Portscans, TCP-Portscans, OS Fingerprinting)

TCP Spoofing/Hijacking

Ist möglich aber schwierig. Wenn Knoten A Knoten B vertraut, kann versucht werden B auf TCP-Level zu imitieren. Der Angreifer muss dabei eine getückte TCP-Anfrage schicken, das Opfer DoSen und die korrekte sequence number erraten oder erschnüffeln. Das alles muss während des 3-Weg-Handschlags passieren. Es ist einfacher Attacken auf Layer 3 oder 2 (IP, ARP) auszuführen.

TCP DoS Attacken

SYN Flooding Attacke: Angreifer fängt mit Verbindungsherstellung an; Opfer allokiert Speicher für die Verbindung (reassembly buffer, ...); Angreifer antwortet nicht mehr ➔ halb offene Verbindung bleibt auf Opferseite bestehen; ist die maximale Anzahl an Verbindungen erreicht, kann das Opfer keine neuen Verbindungen mehr entgegennehmen.
Lösung: SYN Cookies; Halb offene Verbindungen in FIFO-Reihenfolge schließen

Process Table Attacke: Dämonen sind Programme die an einem Port auf Verbindungen warten und diese dann einem Thread zuteilen. Viele solcher Dämonen haben root-Reche. Bei vielen Verbindungen wird die Process-Table aufgefüllt und es können keine neuen Verbindungen mehr aufgebaut werden.

2. Internet Applications

Internetinfrastruktur

Traditionell alte Services die zu den Anfängen des Internets gebraucht und entwickelt wurden. Bietet Basisservices an die von anderen Applikationen genutzt werden. Oftmals sind wenige oder gar keine Sicherheitsmechanismen implementiert, da zur Zeit der Erschaffung vieler Protokolle davon ausgegangen wurde, dass nur vertrauenswürdige Mitarbeiter Zugang zum Internet hätten.
Beispiele für Services: remote access (ssh), name resolution (DNS), file transfer (FTP), mail transfer (SMTP), time synchronisation (NTP)

Remote Access

telnet und rlogin: grauenhafte Sicherheit; Klartextpasswörter, connection hijacking...
ssh: version 1: unsicher, da Möglichkeit besteht Daten in entfernten Stream zu bringen; version 2: Port Tunneling, Remote Copy

DNS (Domain Name System)

Mapt Domainnamen auf IP-Adressen mittels verteilter Datenbank; Jede Domain wird durch Nameserver verwaltet; Verwendet hauptsächlich UDP; Root name server sind auf 13 Maschinen auf der Welt verteilt; Es gibt verschiedene Servertypen: primary (verlässlich für die Domain; lädt Daten von Speicher), secondary (backup server, holt Daten durch Zone transfer), caching-only (holt Daten von anderen Servern, cacht Ergebnisse aber), forwarding (gibt alle Anfragen an andere Server weiter) Kann ein Server eine Anfrage nicht beantworten, leitet er sie an einen Server eine Ebene darüber weiter. Ergebnisse werden meistens gecached; Reverse lookup: von IP-Adresse auf Namen mappen

Es muss mindestens ein DNS-Server definiert sein. Anfragen können iterativ oder rekursiv erfolgen:
rekursiv: es braucht einen Nameserver um die Antwort auf die Anfrage zu finden;
iterativ: anstatt eines gemappten Namens antwortet der Server mit der Adresse eines anderen Servers (Answer: I don't know, but ask 1.1.1.1)

DNS Data

Sowohl Anfrage- als auch Antwortnachrichten sind im selben Format (binary); DNS Data ist in Resource Records strukturiert, welche die Informationen beinhalten

DNS Sicherheitsprobleme

Simple DNS Spoofing: wird verwendet wenn Authentifizierung mittels reverse lookup durchgeführt wird (z.B. trusted.example.com darf sich auf rlogin ohne username/passwort einloggen); Durchführung: der Domainname wird durch eine reversed DNS query erhalten → eine DNS-Anfrage wird (unter Kontroller des Angreifers) zum zuverlässigen DNS Server (für die IP-Adresse zum einloggen) weitergeleitet → dieser DNS Server antwortet mit dem (falschen) vertrauten Namen

Gegenmaßnahmen: double reverse lookup verwenden (reverse lookup 66.66.66.66 → trusted.example.com; lookup trusted.example.com → 123.1.2.3 → login verweigern!)

DNS Cache Poisoning: DNS-Anfragen werden normalerweise über UDP gesendet → Antwort kann gespoofed warden (Race condition gegen legitime Antwort)
DNS Hijacking Möglichkeiten: Wettlauf mit dem Server um Client, oder Wettlauf mit Server um anderen Server
Spoofed DNS reply: Muss einer Anfrage matchen! → korrekte (spoofed) Quell-IP-Adresse des echten Servers verwenden; korrekten Ziel-UDP-Port verwenden (=Quellport von dem aus Anfrage versendet wurde); Auf richtige Query antworten; richtigen Wert für DNS nonce Feld verwenden (16 bit zufällige request-ID)

DNS Cache poisoning: Caching-only-server werden angegriffen; Anfrage für host.example.com senden → Server wird Anfrage and Nameserver für example.com senden → viele fake Antworten zu diesem Server schicken (mit der IP-Adresse des NS von example.com als Quell-IP (~4 Möglichkeiten), Ziel-UDP-Port und DNS nonce müssen geraten werden (jeweils 16 Bits) → Durchschnittlich müssen $2^{16} * 2^{16} * \frac{4}{2} = 2^{33} \approx 8$ Millarden Antworten gesendet werden
Um die Chancen zu erhöhen kann versucht werden den Quellport herauszufinden (viele Server verwenden immer denselben UDP-Port) → $2^{16} * \frac{4}{2} = 2^{17} \approx 130\,000$Möglichkeiten;
Birthday attack: manche Server erlauben mehrere ausstehende Anfragen für dieselbe Domain → sende 100 Anfragen + 100 Antworten ~= 10000 Ratemöglichkeiten
Wird das Rennen jedoch verloren, muss gewartet werden bis der Cache abläuft (~1 Woche)

Effekte des DNS Cache poisoning: der Traffic kann umgeleitet werden; DoS; MITM ohne physischen Zugang; Umleiten von Emails; auto-updates ausnutzen (java updater nutzt keine Verschlüsselung → poison java.sun.com)

DNS Level Poisoning wird nicht nur von den „bösen“ verwendet (viele ISP verwenden es).

Gegenmaßnahmen

Verwendete DNS-Server checken (zufällige Quellports verwenden); ausgehenden Anfragentraffic sniffen (oft nicht möglich); Blockiere Anfragen zu rekursiven Resolvern von außerhalb des eigenen Netzwerks.

FTP (File Transfer Protocol)

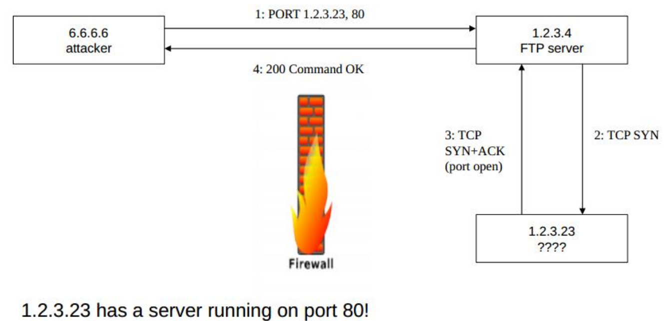
Basiert auf TCP; Client/Server Architektur: client (ftp) sendet eine Verbindungsanfrage an Server (ftpd); Server lauscht auf Port 21; Client schickt Usernamen und Passwort zur Authentification; Client verwendet GET und PUT Befehle um Dateien zu verschicken
Es werden 2 TCP-Verbindungen verwendet: eine für die Kommandos, und eine zweite um die Daten zu übertragen
2 Modi: Active Mode: Client weist Server an sich mit einem seiner lokalen Ports zu verbinden (mit dem PORT-Befehl); Server öffnet Verbindung von Port 20 zu zuvor spezifizierten Port; Transfers finden statt und die Verbindung wird geschlossen
Passive Mode: Client verwendet PASV-Befehl; Server öffnet einen Port und sendet dessen Nummer zum Client; Client verbindet sich mit dieser Portnummer; Transfers finden statt und die Verbindung wird geschlossen

FTP Security

Konfigurationsfehler: anonyme User dürfen Dateien schreiben; in home directory schreiben: kann ausgenutzt werden um Dateien die zur Authentifikation genutzt werden zu bearbeiten (ssh/authorized_key; ...)

PASV Connection Theft: Angreifer kann versuchen sich mit dem geöffnetem Port vor dem Client zu verbinden; Da die Verbindung für die Befehle noch aufgebaut ist, verursachen die Clientbefehle Transfers zwischen dem Server und dem Angreifer

FTP Bounce: Der PORT-Befehl gibt dem Server die Adresse die er zur Öffnung einer Verbindung verwenden soll. Laut Definition muss diese Adresse jedoch nicht mit der des Clients übereinstimmen ➔ dem Server kann der Befehl erteilt werden eine Verbindung zu einem dritten Client aufzubauen; kann verwendet werden um TCP Portscan durchzuführen (damit kann sogar hinter eine Firewall gescannt werden, da es scheint, dass der Server die Anfrage schickt).



1.2.3.23 has a server running on port 80!

Kann auch verwendet werden um Daten an Ports zu schicken (dadurch können IP-basierte Authentifizierungen umgangen werden): Hochladen der Daten auf den ftp-Server (PUT mydata) ➔ PORT Ziel-IP, Ziel-Port ➔ GET mydata

SMTP (Simple Mail Transfer Protocol)

De facto Standard für Emailübertragung; Simples textbasiertes Protokoll; Port 25;
push Protokoll: um Emails zu senden oder zwischen Servern weiterzuleiten; Clients müssen Emails mit anderen Protokollen holen (IMAP oder POP)
SMTP verwendet keine Authentification: jeder kann sich mit einem SMTP-Server verbinden und Nachrichten senden; der Server kann dabei nur die IP-Adresse/den Domain Namen feststellen; Grund hierfür ist, dass Emails ein offenes, verteiltes System sind (jeder kann jedem Emails schicken; keine zentrale Autorität)

Der Mailserver example.com sollte nur Nachrichten von/nach Emailaccounts example.com ausliefern
SMTP Authentication: Überprüfen ob User in example.com Netzwerk vorhanden; Erweiterungen:
SMTP-AUTH: Zugangsberechtigungsliste mit explizitem Login (Clients müssen SMTP-AUTH kennen)
POP-before-SMTP: logins werden durch POP-Anfragen simuliert (benötigen Logins). Wenn ein Client eine POP-Anfrage durchführt, wird seine IP-Adresse für einige Zeit (z.B. 30 Minuten) auf dem SMTP-Server freigeschaltet

MTA (Mail Transfer Agent)-Encryption: POPS/IMAPS/SMTPS (SSL/TLS)

Erweiterungen: STARTTLS: verwendet Klartext-Standard-Ports, kann aber mittels STARTTLS-Befehl verschlüsselte Session starten
Dedicated SSL Ports: POP 995, IMAP 465 und SMTP 993 verwenden verpflichtend Verschlüsselung

Address Spoofing

Authentifikation durch IP: jeder in example.com kann emails von ceo@example.com senden
Sender kann Quelladresse fälschen: kann behaupten email von mybank.com zu sein

Gegenmaßnahmen

SPF (Sender Policy Framework): aushebeln der DNS Infrastruktur: Besitzer von mydomain.com spezifiziert welche IPs autorisiert sind emails von *@mydomain.com zu senden. Verwendet TXT-records in DNS-Servern (➔ keine Änderungen in DNS Implementierung nötig); SMTP-Server kann IP/Domain Namen checken

Spam

Versucht email-Adressen ausfinding zu machen (bruteforce, im Web suchen, Stehlen durch Maleware, ...); Versendet werden Spams durch open proxies, web forms, zombie nets, ...
Gegenmaßnahmen: Spamfilterwerkzeuge, Blacklists, Greylists (temporärer Ausschluss von unbekannten Absendern; echte Sender werden es später nochmals versuchen), Infrastruktur (SPF)

NTP (Network Time Protocol)

Mit eigener oder höherer Ebene wird Zeit synchronisiert; niedrigeren Schichten wird Zeit angesagt

DoS & DDoS

Reflected DoS: Opfer sieht nur Pakete des Reflectors, Reflector sieht nur gespoofte Pakete; hoher Amplifikationsfaktor

3. General Unix Security

Der Kernel bildet eine Hardwareabstraktionsschicht für User-Space-Programme. Er hat kompletten Zugriff auf alle (physischen) Ressourcen. Services stehen mittels System calls zur Verfügung.
System calls führen einen Übergang vom user mode zum kernel mode durch.
Kernelschwachstellen beeinträchtigen normalerweise das komplette System. Attacken werden mittels System calls durchgeführt.
Problematisch sind vor allem Gerätetreiber, da diese oft im Kernelmode ausgeführt werden.
Code der im User Mode läuft ist immer zu einer bestimmten Identität gelinkt. Unix ist userzentrisch (es gibt keine Rollen). Ein User wird durch seinen Usernamen (UID) und seinen Gruppennamen (GID) identifizert und durch sein Passwort authentifiziert.
Prozesse implementieren user-activities; haben (im Gegensatz zu Threads) eigene Memory Pages und File Descriptor Tabellen.
Shadow Passwords: Die Passwortdatei (/etc/passwd) wird von vielen Programmen verwendet um User ID mit Usernamen zu mappen ➔ sie muss durch den User lesbar sein. Verschlüsselte Passwörter sind in /etc/shadow gespeichert und können nur durch Superuser und priviligierte Programme gelesen werden.
Jeder User gehört mindestens einer Gruppe an. Zugriffsrechte auf Dateien und Ordner werden mit Permissonbits geregelt (chmod, chown, chgrp, unmask).

Type	r	w	x	s	t
File	read access	write access	execute	suid / sgid inherit id	sticky bit
Directory	list files	insert and remove files	stat / execute files, chdir	new files have dir-gid	files only delete-able by owner

Jedes Programm hat eine reale und eine effektive user/group ID die normalerweise identisch sind. Real ID: wird durch aktuellen User festgelegt; effective ID: geben die Rechte des Prozesses an (suid/sgid Bits)
Kommunikation zwischen Shell und Programmen über Redirictions und Pipes.

Shell Attacks

Die Umgebungsvariablen $HOME und $PATH können Verhalten von Programmen die mit relativen Pfadnamen arbeiten verändern. Der Interne Feld Seperator ($IFS) wird verwendet um Tokens zu trennen (wenn auf „/“ gesetzt wird „/bin/ls“ zu „bin ls“ ausgewertet).

Preserver attack: vi ruft /bin/mail auf wenn er crasht um die Datei zu sichern. IFS ändern und bin als Link zu /bin/sh erstellen und vi crashen lassen ➔ /bin/sh wird ausgeführt

Kontroll- und Escape Zeichen: müssen vor dem Input überprüft werden (sonst können ‘;‘, ‘&‘, usw. vom Benutzer eingefügt werden); Restricted shell (mit –r) hat ein kontrollierteres Umfeld

system(char *cmd): exekutiert Kommandos; kann binäre Programme angreifbar machen (vor allem jene die Userinput verwenden)

File Descriptor Attacken

SUID Programm öffnet Datei und forked externen Prozess. Wenn das close-on-exec-Flag nicht gesetzt ist, erbt der neue Prozess den Filedescriptor. Manche Angreifer nutzen diese Schwachstelle aus.

Signals

Signals sind eine einfache Form für Unterbrechungen. Sie sind asynchron und können überall in Prozessen im user space passieren. Bis auf SIGKILL können alle Signals durch einen Prozess abgefangen werden (signal handler). Ist kein Signalhandler implementiert, so wird das Signal entwedet ignoriert oder der Prozess wird gekillt.
Wegen Signals muss Code re-entrant sein (atomare Modifikationen, keine globalen Datenstrukturen); es können Race Conditions entstehen. Secure Signals: Handler sind so einfach wie möglich zu implementieren. Signals im Handler sollen geblockt werden und es sollen nur asynchronsichere Funktionen aufgerufen werden.

Shared Libraries

Static shared library: Adresse wird zur Linkzeit gebunden → unflexibel wenn sich Library ändert, dafür schneller
Dynamic shared library: Adresse wird zur Ladezeit gebunden → der Code wird langsamer ausgeführt

4. Web Security

Webapplikationen sind Programme die auf einem Server laufen, Input von außerhalb (durch das Internet) empfangen, verarbeiten und eine Antwort zurücksenden.
Piggypacking: Attacken, die in erlaubten HTTP-Requests versteckt sind, werden von Firewalls oft einfach durchgelassen.

HTTP (HyperText Transfer Protocol)

Wird verwendet um mit Webapplikationen zu kommunizieren. HTTP-Transaktionen bestehen aus 3 Teilen: (1) Request- oder Responsezeile (GET, OK, ...), (2) Headerdaten (Host, User-Agent, Content-Type,...), (3) Entity-Body (eigentlicher Inhalt, z.B. HTML)
Über GET werden Parameter direkt in URL encodiert, bei POST werden sie im Entity-Body mitgeschickt.
Da HTTP meist nicht ausreicht, werden oft auch Scriptsprachen auf dem Server implementiert.

Selbst Intranetapplikationen können von außen gefährdet sein. Bösartige Inhalte die über das Web verschickt werden, können interne Knoten zu Angriffen auf das Intranet bringen. Dagegen helfen Regeln, die auch internen Usern den Zugriff auf Applikationen verweigern.

OWASP (Open Web Application Security Project)

Top vulnerability list:

1. Injection: wenn nicht vertraute Daten als Kommando oder Abfrage zu einem Interpreter geschickt werden (z.B. SQL, OS (shell), ...). Die Daten werden dann als Kommandos interpretiert
2. Broken Authentication and Session Management: Applikationsfunktionen für die Authentifizierung und für die Sitzungsverwaltung sind oft nicht korrekt implementiert, wodurch Angreifer versuchen können Passwörter, Schlüssel, Session Tokens usw. zu stehlen und die Identität von Nutzern erfahren können.
3. Cross Site Scripting (XSS): wenn Applikationen fremde Daten an den Webbrowser senden ohne sie davor zu validieren. Dadurch können Angreifer Scripts in den Browsern der Opfer ausführen.
4. Insecure Direct Object References: Wenn eine Referenz auf ein internes Objekt (Datei, Ordner, Datenbankschlüssel, ...) öffentlich gemacht wird. Sind keine Zugriffs-schutzmechanismen implementiert, können Angreifer diese Referenz manipulieren und auf nicht autorisierte Daten zugreifen.
5. Security Misconfiguration: Falsche Sicherheiteskonfigurationen (z.B. Standardpasswörter und Usernamen verwenden, ungenützte Features aktiviert lassen, ...).
6. Sensitive Data Exposure: Viele Webapplikationen schützen sensible Daten (Kreditkarten-informationen, ...) nicht ausreichend mit Verschlüsselung oder Ähnlichem.
7. Missing Function Level Access Control: Applikationen schützen die Funktionen nicht immer ausreichend.
8. Failure to Restrict URL Access: Viele Webapplikationen prüfen die URL-Zugriffsrechte bevor geschützte Links und Buttons gerendert werden. Diese Prüfung sollte allerdings auch passieren bevor eine versteckte Seite aufgerufen wird.
9. Insufficient Transport Layer Protection: Applikationen verschlüsseln den Netzwerktraffic oft garnicht oder nicht ausreichend
10. Using Components with known vulnerabilities
11. Unvalidated Redirects and Forwards
12. Buffer Overflows
13. Improper Error Handling: Wenn ein Angreifer es schafft einen Fehler hervorzurufen der nicht korrekt gehandhabt wird, kann er detailierte Systeminformationen herausfinden, den Service verweigern oder den Server crashen lassen.
14. Denial of Service
15. Unvalidated Input

Unvalidated Input

Webapplikationen verwenden HTTP-Anfragen um zu eruieren wie sie antworten sollen. Angreifer können versuchen teile der HTTP-Anfrage (inklusiver der URL, dem Header, den Cookies, Felder, ...) zu manipulieren um Sicherheitsmechanismen zu umgehen (oft verwendet: XSS, SQL-Injection, hidden field manipulation, parameter injection). Der Versuch schadhaften Input rauszufiltern ist schwierig, da es viele Wege gibt Informationen zu codieren.
Oft wird eine clientseitige Validierung der Daten erwartet. Diese ist jedoch sehr einfach zu umgehen. Es sollte jeder Userinput gecheckt werden. Besser mit Whitelists als mit Blacklists (da hier leicht etwas übersehen werden kann).

SQL Injection

Es sollte sichergestellt werden, dass Nutzer keine unnötigen Debug- und Fehlermeldungen angezeigt bekommen (für das Debuggen eignen sich Logdateien besser). Advanced SQL Injection: wenn ‘ und “ rausgefiltert wird, kann dies mittels char(0x63) umgagnen werden. Blind SQL Injection: es werden keine Fehlermeldungen angezeigt und der Angreifer muss sich vortasten und die Angriffs-möglichkeiten schrittweise herausfinden. Second Order SQL Injection: die SQL Injection wird in eine Applikation eingeschleust, aber sie wird erst später ausgeführt (z.B. Gästebuch). Lösung gegen SQL Injections: Applikation von SQL isolieren und Prepared Statements verwenden.

Hidden Field Tampering

In HTML-Code werden manchmal versteckte Felder für temporäre Daten verwendet. Diese können leicht aufgespürt und verändert werden.

Parameter Injection

Applikationen die Paramter vom Nutzer erwarten und diesen nicht überprüfen sind hierfür anfällig. Lösung: Input immer überprüfen und keine Zeichen erlauben die spezielle Bedeutung für shell haben (| * > < ;).

Session Management

HTTP ist zustandslos (stateless) → es weiß nichts von vergangenen Anfragen. Um dies zu lösen (z.B. damit ein User eingeloggt bleibt) gibt es Sessions. Webapplikationen verwalten die Sessions selbst und speichern sie serverseitig. Wird eine Session erstellt, wird der Client über dessen ID informiert und schickt sie bei jedem HTTP-Request mit. Werden die Sessiontoken nicht richtig geschützt, können Angreifer sich als fremde Person ausgeben.
3 Möglichkeiten um Session IDs zu übermitteln:

1. In URL als GET Parameter codieren: Nachteil: sichtbar in der URL-Bar, selbst wenn verschlüsselte Übertragungen verwendet werden
2. Als hidden Field: Funktioniert nur bei POST-Requests
3. Als Cookies: Nachteil: kann vom Client nicht akzeptiert werden

Cookies

Cookies sind Tokens die als Key-Value-Paare vom Server clientseitig gespeichert werden.
Non-persistent Cookies: werden nur für die Browsersessiondauer gespeichert
Secure Cookies: werden nur über verschlüsselte Verbindungen (TLS) gesendet
Nur die Cookies zu verschlüsseln (auf unverschlüsselter Verbindung) bringt wenig, da die Cookies gestohlen und wiederverwendet werden können → Abhilfe: Cookies die IP-Adresse beinhalten. Dafür kann die Session unterbrochen werden wenn sich die IP-Adresse des Clients ändert

Session Attacks

Session ID stehlen:
Interception: Request oder Response abfangen und SessionID extrahieren (Schutz indem TLS für jede Request/Response verwendet wird und nicht nur fürs Login)
Prediction: versuchen die SessionID vorauszusagen oder zu erraten (Schutz indem nur ZufallsIDs verwendet werden)
Brute Force
Fixation: Das Opfer dazu bringen eine bestimmte SessionID zu verwenden

Session IDs müssen einzigartig sein, und sollten resistent gegen Brute Force Attacken sein. SessionIDs können gehasht und danach verschlüsselt werden um einen zufälligen Sessiontoken mit Signatur zu erzeugen.

JavaScript

JavaScript wird in Webseiten eingebunden um dynamisches clientseitiges Verhalten zu implementieren. Typischerweise wird es für dynamische Interaktionen (URL eines Bildes ändert sich), clientseitige Validierung (hat User Nummer eingegeben?) und zur Manipulation von Document Object Models (DOMs) verwendet.

Vorteile von JavaScript sind, dass die Benutzerumgebung durch Sandboxing geschützt ist, und JavaScript Programme untereinander geschützt sind. Allerdings bringen diese Schutzmechanismen nichts, wenn der Benutzer dazu gebracht wird schadhaften Code von einer „*vertrauten*“ Quelle herunterzuladen.

Cross Site Scripting (XSS)

Angreifer kann XSS verwenden um schadhafte Scripts an nichtsahnendes Opfer zu schicken. Der Browser kann nicht wissen, ob das Script vertrauenswürdig ist und führt es einfach aus. Da davon ausgegangen wird, dass das Script von einer vertrauenswürdigen Quelle stammt, kann es auf Cookies, Sessiontokens und andere sensible Informationen zugreifen die mit der Webseite verwendet wurden. Es kann sogar der komplette Inhalt einer HTML-Seite neu gestaltet werden (➔ Phishing). Same Origin Policy: Scripts dürfen nur auf eigene Seite zugreifen.

Stored XSS Attacks: der schadhafte Code ist dauerhaft auf dem Server gespeichert. Scripts werden ausgeführt sobald Opfer Website öffnet (z.B. in einem Forum)
Reflected XSS Attacks: der schadhafte Code wird vom Server zurückgegeben und nicht gespeichert; z.B. in einer Fehlernachricht, einem Suchergebnis, oder irgendeiner anderen Antwort die den gesamten Input (oder zumindest Teile davon) beinhaltet. Scripts müssen Opfer zugestellt werden (z.B. über Email) und werden von ihm angeklickt und so zum Server übertragen und zurückgegeben.

Gegenmaßnahmen

Sicherstellen, dass Applikationen alle Header, Cookies, Querystrings, Formularfelder und versteckte Felder auf Regeln überprüfen (mittels Anti-XSS Filtern).
Content Security Policy (CSP): Code und Daten separat halten
Firewalls auf Applikationslevel
httpOnly: Cookieoption die dem Browser mitteilt keine Scriptsprachen auszuführen

Improper Error Handling

Problematisch ist vor allem wenn Endnutzer interne Fehlerinformationen angezeigt bekommen. Fail-open security check: Fehler ist aufgetreten, aber Authentifizierung wurde trotzdem bestätigt. Um dem entgegenzuwirken sollten Fehlerbehandlungen gut dokumentiert werden.

Insecure Storage

Die meisten Webapplikationen müssen sensible Daten speichern. Es passieren dabei oft Fehler beim Verschlüsseln sensibler Daten.

Denial of Service (DoS)

Webapplikationen können nicht einfach feststellen ob eine Anfrage bösartig ist oder nicht, da nicht gewusst wird von wo/wem HTTP-Anfragen stammen.

5. Testing

Validierungstests: ist das System richtig designt?
Verifikationstests: ist das System richtig implementiert?
Whit-Box-Tests: Testen der Implementation (z.B. mittels Zweig-, Bedingungs- und Pfad-überdeckungstests). Versuchen Implementierungsfehler zu finden; kann nicht garantieren, dass Spezifikationen erfüllt wurden.
Black-Box-Tests: Auf Spezifikation testen. Input-Output-Tests. Kann nicht garantieren, dass Implementierung korrekt ist.

Statisches Testen: Anforderungen und Designdokumente prüfen; Quellcode anschauen; es wird kein Code ausgeführt.
Dynamisches Testen: Programme mit Input füttern und Verhalten beobachten
Automatisches Testen: Testen sollte kontinuierlich stattfinden → viele Input- und Output-Vergleiche notwendig → kann oft automatisiert werden
Regressionstests: Überprüfung ob durch neue Funktionalitäten keine alten „verlorengingen“
Software Fault Injection: Bugeffekte anstatt Bugs selber verfolgen; Da Bugs nicht komplett entfernt werden können, sollen Programme Fehlertolerant sein; Fehler werden absichtlich im Code eingefügt und die Effekte werden beobachtet um das Programm robuster zu machen

Security Testing

Viele Testverfahren können auch genutzt werden um Sicherheitsprobleme festzustellen. Testen sollte in allen Phasen des SW-Zyklus stattfinden.

Phase der Anforderungsanalyse

Systemanforderungen beinhalten in dieser Phase meist nur funktionale Anforderungen → Sicherheitsanforderungen werden oft außer Acht gelassen → sind Sicherheitsanforderungen nicht angegeben, werden sie meist auch nicht implementiert und getestet → Programme sind durch Design unsicher → Lösung: Von Anfang an spezifizieren wie System auf Attacken oder in Ausnahmefällen reagieren soll

Designphase

Nicht viele Werkzeuge verfügbar → manuelle Designreviews (formale Methoden, Attackgraphen)
Formale Methoden: formale Spezifikationen die mathematisch beschrieben und geprüft werden können; wird für kleine, sicherheitskritische Anwendungen verwendet; Zustände und Zustands-änderungen müssen formalisiert werden und ein Modelchecker schaut, dass keine unsicheren Zustände erreicht werden können
Attackgraphen: gegeben: Zustandsmodell M und Sicherheitsmerkmale P → eine Attacke ist eine Ausführung von M, sodass P gebrochen wird. Attackgraphen können per Hand oder automatisch erfolgen

Implementierungsphase

Hier gibt es mehr Werkzeuge um bekannte Probleme und Bugs aufzudecken. Testen zielt auf bestimmte Schwachstellen ab. Mittels statischer und dynamischer Analyse möglich.

Statisches Sicherheitstesten

Manuelle Überprüfung (z.B. auf Kommentare, funktionelle Zusammenfassung jeder Methode, ...): mühselige und schwierige Aufgabe

Syntax-Checker: Quellcode parsen und problematische Funktionen überrüfen (strcpy()); kann komplexere Beziehungen nicht verstehen (limitierter Support für Argumente); keine Kontroll- oder Datenflussanalyse möglich

Annotationsbasierte Systeme: Eigenschaften werden im Code mittels Annotationen beschrieben (@NotNull); Analysewerkzeuge prüfen den Code auf mögliche Eigenschaftsverletzungen; Probleme sind generell nicht entscheidbar ➔ Kompromiss zwischen correctness und completeness

Model checking: Programmierer spezifiziert Sicherheitseigenschaften die eingehalten werden müssen; Modelle sind als Zustandsmaschinen dargestellt; mittels Kontroll- und Datenflussanalyse wird überprüft ob keine unsicheren Zustände erreicht werden.

Meta-compilation: Programmierer fügt einfache systemspezifische Compilererweiterungen ein. Diese prüfen (oder optimieren) den Code; kann viele Bugs finden (allerdings nur wenig komplexe); Model-Checking ist schwieriger, aber besser wenn es einmal konfiguriert wurde.

Dynamisches Sicherheitstesten

Überprüfung zur Laufzeit: zwischen BS und Programm: Systemcalls werden abgefangen und überprüft; zwischen Libraries und Programm: Libraryfunktionen werden abgefangen und überprüft, wird oft verwendet um Speicherprobleme und Bufferoverflows zu finden

Profiling: Loggen des dynamischen Verhaltens einer Applikation

Fuzz testing (fuzzing): um Brute-Force-Schwachstellen zu entdecken; füttert Programm mit großer Menge von (semi-) zufäligem Input; Programm wird auf Deadlocks, Abstürze, usw. beobachtet; für das Auffinden von Protokoll/Datei-Parsingfehler gut geeignet; Fuzzer hält sich an Spezifikationsregeln

(Pre-) Rollout Phase

Code für Auslieferung vorbereiten: Debugcode, sensible Informationen, Testaccounts, usw. entfernen; alle Sicherheitseinstellungen zurückstellen

Penetration Testing: Prozess zur aktiven Evaluierung der Informationssicherheitsmaßnahmen (ähnlich wie InetSec Challenges); Analyse der Designschwachstellen sowie technischen Fehler und in Report festhalten

external penetration testing: traditionelle Art: Fokus auf Services und Verfügbarkeit von Servern

internal security assessment: testen über LAN, DMZ, Netzwerkpunkten

application security assessment: Applikationen die sensitive Daten offenbaren könnten werden getestet

wireless/remote access assessment: wireless access points, Konfigurationen, ...

telephony security assessment: mailbox-Sicherheit, ...

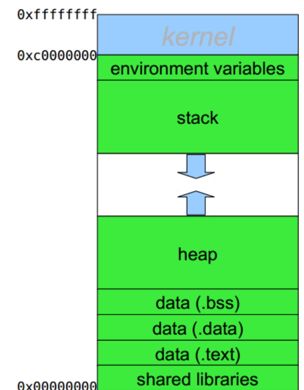
social engineering: piggypacking, Telefonate, ...

Grenzen von Penetration Testing: Die Erlaubnis anzugreifen: Der Kunde gibt an in welchem Ausmaß welche Systeme wann getestet werden dürfen; Report schreiben: Kunde zahlt für Report (was viel Zeit in Anspruch nimmt)

6. Memory Corruption (Buffer Overflows)

Buffer Overflows sind eine der häufigsten Attacken auf Systeme. Ist die Attacke erfolgreich, lässt sich schadhafter Code auf dem System ausführen. Schritte für Buffer Overflow:
1) Instruktionen in Speicher des anfälligen Programms einschleusen
2) Programmschwachstelle ausnutzen um Kontrollfluss zu ändern
3) (schadhaften) eingefügten Code ausführen

Buffer Overflows sind sehr effektiv und können sowohl lokal als auch von einem entfernten Gerät aus ausgeführt werden. Sie sind jedoch architektur- und betriebssystemabhängig und es müssen Adressen erraten werden. Viele Prorammiersprachen (wie Java) haben einen automatischen Schutz gegen BOs implementiert.

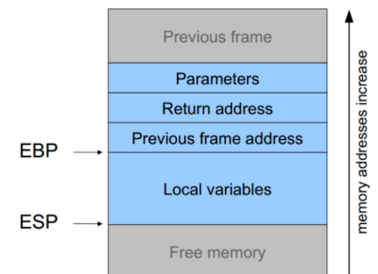


Memory Layout

Stack: beinhaltet lokale Variablen und Prozeduren
Data: globale nicht initialisierte (.bss) und initialisierte (.data) Variablen sowie dynamische Variablen (heap)
Code (text-Segment): Programminstruktionen (normalerweise read-only)

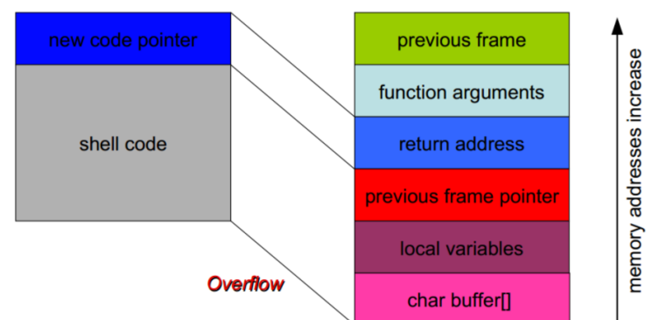
Stack

Der Stack wächst normalerweise in richtung kleiner werdenden Speicheradressen. Der Stack Pointer zeigt auf den obersten Stackeintrag. Wird eine Funktion aufgerufen, kommt ein neues Frame auf den Stack. Beim Return wird Frame wieder vom Stack entfernt. Die Adresse des aktuellen Frames ist im Prozessorregister gespeichert.



Buffer Overflow

Es werden zu viele Daten in ein Array geschrieben, wodurch angrenzender Speicher überschrieben wird (strcpy, strcat, gets, fgets, sprintf, ...). Dadurch können der Code Pointer oder die Applikationsdaten verändert werden. Wird der Code Pointer bewusst auf den eigenen Code gesetzt, so springt der Prozess nach der Abarbeitung auf diesen (schadhaften) Code. Der Code wird dadurch mit den Rechten des Programms ausgeführt. Der Code Pointer muss mit der richtigen Adresse überschrieben werden (ältere BS verwenden für alle Programme selbe Stackstartadressen).



Shellcode

Normalerweise soll eine Shell gestartet werden (mittels execve (=System Call)). System calls sind Mechanismen um Services vom BS anzufordern.
Problem: Die Position des Shellcodes im Speicher ist nicht bekannt → Instruktionen mit relativen Adressen verwenden (jmp oder call); NOP (no operation) sleds (0x90) können am Anfang des Codes verwendet werden → die Rücksprungadresse muss nicht mehr so genau erraten werden.

Ist der Buffer zu klein um den (schadhaften) Code zu beinhalten, kann der Code auch in Umgebungsvariablen gespeichert werden (diese werden auf dem Stack gespeichert). Die

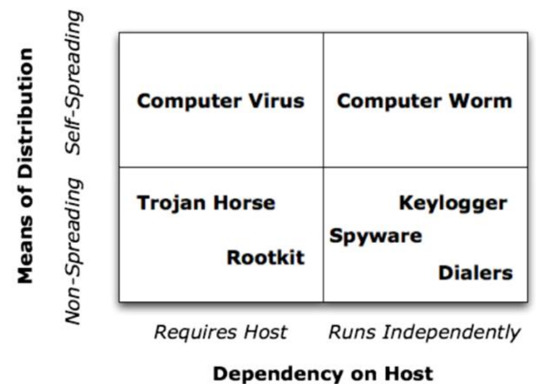
Returnadresse muss dann auf die Umgebungsvariable umgeleitet werden. Dadurch kann der Code auch sehr lang sein aber es muss Zugriff auf die Umgebungsvariablen möglich sein.

Gegenmaßnahmen

Compiler und Linker können manche Schutzmechanismen aktiviern um Buffer Overflows schwieriger (oder gar unmöglich) zu machen: nicht ausführbare Stacks, zufällige Adressenvergabe, write XOR execute, ...; Weiters sollten Funktionen aus C Libraries bei denen bekannt ist, dass sie das Risiko eines Buffer Overflows bergen, vermieden werden (strncpy statt strcpy verwenden).
Address Space Layout Randomization (ASLR): für jede Ausführung eines Programms schaut das Memorylayout anders aus ➔ Rücksprungadresse kann nur mehr sehr schwierig erraten werden (selbst mit NOP sleds); Auf 32-bit Systemen kann mit Brute-Force die Adresse erraten werden indem einfach so oft probiert wird bis es passt

7. Malware

Malware (Malicious Software) beschreibt jegliche Software die einem schadhaften Zweck dient. Ein Virus ist ein Programm, das sich selbst vervielfältigt indem es sich in andere Dateien einschleust. Interaktionsbasierte Würmer verbreiten sich selbst über ein Netzwerk (z.B. über Email) und müssen durch einen User ausgeführt werden (z.B. indem eine exe-Datei geöffnet wird). Prozessbasierte Würmer verbreiten sich selbst und führen sich auch selbst aus (ohne Hilfe eines Users).



Gründe für die Verbreitung von Maleware sind: Mischung von Daten und Code, Homogene Betriebssysteme, nichtsahnende User, finanzieller Erfolg, ...

Virus

Phasen: reproduzieren, infizieren, ausführen
Reproduktionsphase: Virus versucht nicht entdeckt zu werden
Infektionsphase: schwierig vorauszusagen wann Infektion stattfinden wird; viele Viren bleiben im Speicher ansässig; Boot-Viren (im MBR); File infectors: Virus nistet sich in Datei ein (parasitic: Code wird an Datei angehängt und Programm-entry-point verändert; cavity: Code wird in ungenützte Regionen injiziert); Entry Point Obfuscation: Virenscanner testen Programme oft nur auf Entry Point ➔ Code woanders plazieren und Virus erst später ausführen (wenn Programm bereits läuft); Code Integration: Virus und Programm werden zusammengeführt (dafür muss Programm decompiliert werden)
Ausführungsphase: der eigentliche Schaden wird ausgeführt (Daten löschen, ...)

Wurm

Verschiedene Methoden um neue Opfer zu finden:
Email harvesting: Adressbücher, Dateien, Cache, ... durchsuchen
Network share enumeration: Über Windows sind andere Computer im lokalen Netzwerk sichtbar; manche Würmer greifen alles an (auch Drucker, Router, usw.)
Scanning: zufällige IP-Adressen ausprobieren (oder Brute Force)

Exploit-Based Worms: verbreiten sich ohne menschliche Interaktion (über Netzwerkservices) → breiten sich sehr schnell aus

Botnets

Um den Profit von infizierten Systemen zu maximieren, müssen diese kontrollierbar sein (Command & Control aka C&C aka C2). So können zu einem gewollten Zeitpunkt alle infizierten Systeme DDoS-Attacken ausführen, SPAM verschicken, Identitäten stehlen, oder Änliches.
Anforderungen für C&C sind: Flexibilität (Updates), Skalierbarkeit (potentiell millionen Hosts), Stabilität gegenüber Abschaltungsversuche, Schwierigkeit Ursprung der Kommandos nachzuverfolgen

Ein Bot ist ein infiziertes System, das durch einen Angreifer von außen gesteuert werden kann. Verschiedene Arten: Remote control facility: erlaubt dem Angreifer das Sysem komplett zu steuern
Implementation of different commands: bestimmte Befehle können entgegengenommen werden

Bootkis: (Boot + Rootkit) infizieren startup Code (z.B. MBR) → werden vor dem BS-Kernel geladen → können Kernelfunktionalitäten verändern

Häufige Befehle moderner Bots: Email-Adressen sammeln, Keylogger, Netzwerk sniffen, Screenshots anfertigen, HTTP-Server starten, Prozesse killen, Windows-CD-Keys-stehlen, Dateien herunterladen, Shell-Befehle ausführen, Update durchführen (um neue Befehle einzuführen), ...

C&C (Command and Control)

Sind sehr flexibel, aber auch das schwächste Glied im System (es reicht den C&C zu entfernen und alle Bots, die er kontrollierte sind unbrauchbar). Allerdings lassen echte Botnets mehrere Master zu.

Zentralisiere Ansätze

Push style (z.B. IRC); Pull style (z.B. HTTP); Adresse der C&C server müssen für jeden Bot ersichtlich sein (in binär, Konfigurationsdatei, ...). Große Botnets sind oft in kleinere Unterteilt → man kann durch einen einzelnen Bot nicht auf das gesamte Botnetz schließen
IRC (Internet Relay Chat): Standardprotokoll der ersten Botnets; wird heute kaum mehr verwendet
HTTP: ausgehendes HTTP ist überall erlaubt; schwierig normalen HTTP-traffic von ungewöhnlichem zu unterscheiden

Fast Flux Hosting

Technik um zuverlässige Services mit unzuverlässigen Bots zu garantieren; Standorte von Webservern können damit verschleiert werden; infizierte Computer werden zu Providern von schadhaften Inhalten; kann für das Hosten von Master Server verwendet werden

Domain Generation Algorithms (DGA)

Bot Master registriert auf reglulärer Basis neue Domains. Die Bots generieren Listen von Domains und checken ab ob sie neue C&C server sind, und nehmen dann Befehle von ihnen entgegen.
DGA wird oft als Backupmechanismus für C&C verwendet: es werden normale C&C Server verwendet und auf DGA zurückgegriffen falls alle Server down sind. Um ein DGA-Botnet zu beseitigen müssen alle Domains registriert werden (während es für einen Angriff reicht, dass eine einzige Domain registriert wird).

Dezentralisierte Ansätze:
Über P2P ➔ robuster; Botmaster schwieriger zu finden; schwieriger zu beseitigen, da kein Single Point of Failure; Jeder Bot hat Liste von Nachbarn von denen er Befehle entgegennimmt oder sendet

Gegenmaßnahmen
Master beseitigen: ISP anfordern IP-Adresse zu blockieren (Master können allerdings von verschiedenen Providern auf der ganzen Welt verteilt sein; außerdem müssen alle Master gleichzeitig entfernt/blockiert werden, da ein einzelner den Bots neue C&C Server mitteilen kann) Master übernehmen: Kontrolle über IP/DNS-Namen des Masters übernhemen ➔ Befehle an Bots erteilen sich zu deinstallieren oder den Nutzer zu informieren, dass er infiziert ist; kann rechtlich problematisch sein (z.B. wenn unabsichtlich Schaden verursacht wird); funktioniert nur wenn C&C nicht gut geschützt ist (durch Kryptographie)

Sinkholing: DNS-Server geben falsche Informationen weiter (zu nicht-existierenden IPs); alle DGA-generierten Domains von Conficker wurden beispielsweise gesinkholed
P2P Takedown: Jeder Bot hat Liste mit Peers von dem er Befehle akzeptiert / an die er Befehle sendet ➔ alle Einträge dieser Liste sinkholen damit keine Befehle vom Botmaster mehr erteilt werden können (allerdings verwenden die meisten Botnets authentifizierte Kommandos ➔ falsch signierte Befehle werden ignoriert)

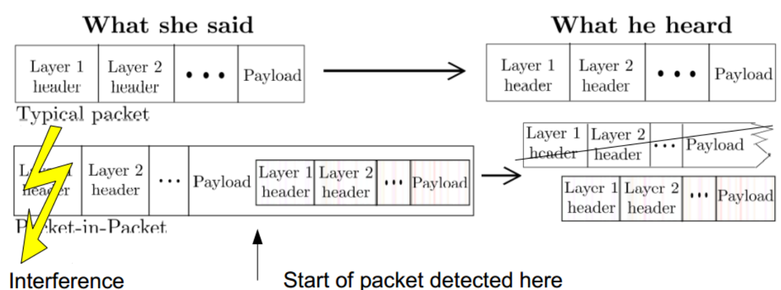
8. Language Security

Postel's Law: “Be conservative in what you send, be liberal in what you accept”

Polyglot: Quellcode der in mehreren Programmiersprachen gültig ist.
Binary Polyglot: Datei die als mehrere Formate gültig ist (z.B. JPEG/PDF/ZIP)

Protokolle sind Sprachen: haben strukturieren Input; Grammatik; werden einer Maschine eingefügt die sie parsen, oder sich nach dessen Anweisungen verhält; Parse ➔ Validate ➔ Use; Protokolle sind oft mächtiger als viele Programmierer denken; Ausnutzung erfolgt oft durch unerwartete Berechnungen aufgrund spezieller Inputs; input recognition == halting problem; ist ein Protokoll zu mächtig, ist seine Validierung im Allgemeinen unentscheidbar

Layer Model
Schichten: kapseln, schützen und sind Black Boxes
Pakete: Sender und Empfänger sind kompatible Geräte; Empfänger liest was Sender übermittelt hat; Störungen (Noise) werden durch niedrigere Schichten gehandhabt



Compiler

Manche Optimierungen die durch Compiler ausgeführt werden sind problematisch: Dead storage elimination: „Write once, read never"-Operationen werden entfernt; Inline functions (Stackframes werden zusammengefügt wodurch private Variablen sichtbar werden können); Alignment on Stack (verringert Entropie des Stacks)

9. Crypto

Shamir's Law: Crypto is bypassed, not penetrated

Kryptographische Ziele:

- Vertraulichkeit (Confidentiality): Inhalt vor unautorisierem Zugriff schützen (Nachrichten lesen)
- Integrität: Inhalt for unautorisierter Manipulation schützen (Nachrichten verändern)
- Authentifizierung: Identifikation von Daten oder Nutzern (unautorisierter Systemzugriff)
- keine Zurückweisung (non-repudiation): Entitäten hindern vorhergehende Aktionen zu leugnen (Signaturfälschung von Dokumenten oder Nachrichten)

Unkeyed Primitives: Hashfunktionen, Zufallssequenzen
Symmetric-key primitives: Signaturen, symmetrische Verschlüsselungen, Authentifizierungscodes für Nachrichten
Public-key primitives: öffentliche Schlüssel Verschlüsselungen, Signaturen

Kerckhoff's principle: Ein Kryptosystem sollte so sicher sein, dass selbst wenn alles über das System (bis auf den Schlüssel) bekannt ist, es immer noch sicher ist (no security through obscurity!).

- Unconditional Security: gegen jeden Angreifer sicher; der verschlüsselte Text birgt keinerlei Informationen über den Klartext (One Time Pad)
- Computational Security: sicher gegen „rechnerisch begrenzte" Gegner; wenn N=NP ist, ist das nicht mehr gegeben (DES, AES)
- Provable Security: sicher gegen „rechnerisch begrenzte" Gegner; es existieren mathematische Beweise die sagen, wie schwierig das Problem zu lösen ist (RSA, DH)

Attacken

- Ciphertext only (COA): Angreifer kennt nur c
- Known plaintext (KPA): Angreifer kennt m und c
- Chosen plaintext (CPA): Angreifer kann m aussuchen und erhält c (Encryption black box)
- Chosen ciphertext (CCA): Angreifer kann c aussuchen und erhält m (Decryption black box)

Passive Attacken beobachten nur, während aktive eingreifen.

Symmetric-Key Cryptography

Gegeben: Alphabet A, Message space M (Elemente aus M sind Klartextnachrichten m), Ciphertext space C (Elemente aus C sind verchlüsselte Nachrichten c), Key space K (Elemente aus K sind Schlüssel k)

Stream ciphers

Spezialfall von Block ciphers mit t=1 (RC4, Salsa20)

Block ciphers

Klartext wird in Stringblöcke fixer Länge t unterteilt. Jeder Block wird einzeln ver-/entschlüsselt (AES, DES, ...); verwendet Substituierungs- und Transponierungstechniken; oft durch mehrere Runden mit verschiedenen Techniken realisiert; versucht Unordnung und Diffusion (Streuung) zu erzielen.

Electronic Code Book (ECB)

Nachricht in Blöcke unterteilen. Nachrichten mit Zufallsdaten füllen, damit sie eine vielfache Länge der Blockgröße haben. Jeden Block einzeln der Verschlüsselungsfunktion $E(k, m[i])$ übergeben (AES)

Passive Attacken:
COA: es kann erkannt werden ob selbe Nachricht nochmals verschlüsselt wurde
KPA: Ist der Klartext bekannt, kann mit demselben Schlüssel immer auf den verschlüsselten Text gemappt werden (passive Attacke)

Aktive Attacken:
COA: Angreifer kann Anordnung der Blöcke ändern, sie löschen oder duplizieren
KPA: Blöcke so verändern, dass sie nach der Verschlüsselung dem Ziel entsprechen
Aktive Attacken sind einfacher wenn CPA möglich ist

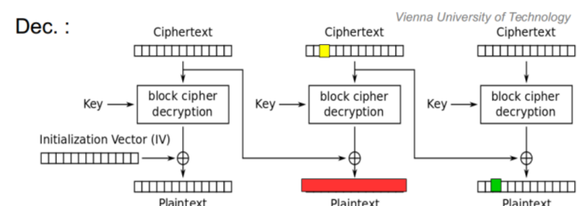
Potentielle Probleme:
Jeder Block wird unabhängig von den anderen verschlüsselt → Datenmuster werden nicht versteckt
Mit selben Schlüssel ergeben selbe Klartexte immer denselben verschlüsselten Text
Anfällig für Blockinsertion oder Blockdeletion; Replay Attacken möglich (erneutes Senden von (verchlüsselten) Nachrichten), kann mittels MAC verhindert werden (verschlüsseln, dann MAC!)

Cypher Block Chaining (CBC)

Wie ECB (unterteilung in Blöcke), jedoch ist die Verschlüsselung eines Blocks von den vorherigen Blöcken abhängig; Verschlüsselung des ersten Blocks von zufälligem Initialisierungsvektor (IV) abhängig; jeden Block XOR mit Ciphertext des letzten Blocks nehmen bevor er Verschlüsselungs-funktion übergeben wird (AES)

Passive Attacken:
COA/KPA: ändert sich IV nicht, werden aus gleichen Klartexten gleiche Ciphertexte

Aktive Attacken:
KPA: CBC Bit Flipping Attack:
der Block der das geflippte Byte enthält wird beim Entschlüsseln wirr sein, während das entsprechende Byte im nächsten Block geändert sein wird



COA: CBC Padding Oracle Attack:
Da Nachrichtenlänge immer Vielfaches der Blockgröße sein muss, wird der letzte Block mit Zeichen aufgefüllt. Eines der häufigsten Verfahren hierfür ist PCKS#5. Dabei wird der Block mit gleichen Zeichen aufgefüllt (Zeichen ist abhängig davon wie viel Platz ist (1: 0x01; 2: 0x02, 0x02; 3: 0x03, 0x03, 0x03, usw.).

Um eine CBC Padding Oracle Attacke ausführen zu können, müssen folgende Punkte gegeben sein:

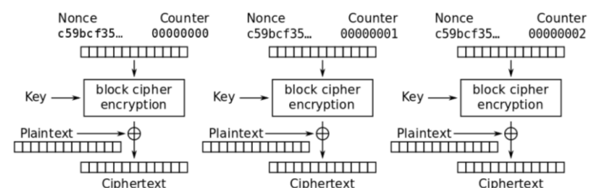
- Wird ein gültiger Ciphertext empfangen, antwortet die Applikation normal
- Wird ein ungültiger Ciphertext empfangen, wirft die Applikation eine kryptographische Ausnahme
- Wird ein gültiger Ciphertext empfangen, dieser aber zu einem ungültigen Wert entschlüsselt, zeigt die Applikation eine Fehlermeldung an

Es wird versucht den Klartext zu bekommen indem der echte IV benutzt wird und der IS (intermediate state) gebruteforced wird ($m = IV \oplus IS$).

Cipher Block Chaining Message authentication code (CBC-MAC):
MAC als Prüfsumme mit Hilfe eines shared secretes erstellen. Damit kann Integrität der übermittelten Daten geprüft werden

Counter Mode (CTR)

Nachrichten mit Zufallsdaten auffüllen (anstatt mit PCKS#5). Verschlüsselung eines Blocks ist abhängig vom aktuellen Counterwert (Counter value). Jeden Block mit dem Output der Verschlüsselungsfunktion XORen um Ciphertext zu erhalten.

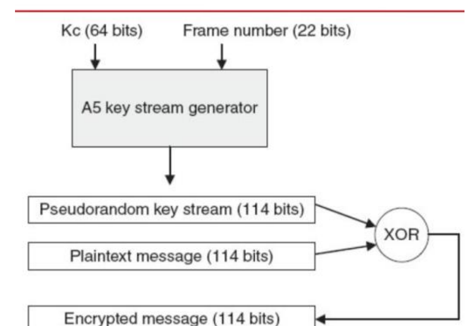


Hashfunktionen

Eine Hashfunktion H nimmt eine Nachricht m entgegen und erstellt daraus einen Hashwert h mit fixer Länge. Eine kryptographische Hashfunktion muss folgende Eigenschaften erfüllen:

- easy to compute: sie muss für jede Nachricht m einfach zu berechnen sein
- pre-image resistance: es ist unmöglich (außer mittels Bruteforce) aus einem Hash eine Nachricht abzuleiten
- second pre-image resistence: es soll so gut wie unmöglich sein bei einer gegebenen Nachricht eine zweite zu finden die denselben Hashwert hat
- colliosion resistance: es soll sehr schwierig sein irgendwelche 2 verschiedene Nachrichten zu finden, die denselben Hashwert haben

10. Mobile Phone Network Security



Broadcast Channels: senden „beacon“ Informationen, sind unverschlüsselt
Dedicated Channels: Kommunikation zu einem bestimmten Benutzer, meist verschlüsselt

IMEI (International Mobile Equipment Identifier): das Handy
IMSI (International Subscriber Identifier): Die SIM-Karte (der Nutzer)
TMSI (Temporary Mobile Subscriber Identifier): temporäre Usuer/Session ID; damit soll Tracking verhindert werden

TMSI deanonymization:

Paging Traffic aufnehmen; bekannte Nummer anrufen und auflegen bevor Channelsetup abgeschlossen

Decryption:

GSM-Cipher kann mittels Rainbowtables (~2TB) geknackt werden.

IMSI Catchers aka Stringray

Wird verwendet um User zu tracken, Texte, Daten und Telefonate abzuhören, MITM, Handys oder SIMs anzugreifen, TANs abzufangen, ...

Identification only: IMSI/IMEI/TMSI wird gestohlen; Tracking; Locationupdates werden abgelehnt
Traffic MITM: Traffic aktiv abfangen; aktive oder passive Entschlüsselung
UMTS downgrade: UMTS-blockieren; Systemnachrichten spoofen
Hold but intercept passively: Imprison in cell, so phone is not lost to a neighbor cell (keine Ahnung?!)

IMSI Catcher erwischen

Ungenützte oder bewachte Channel

11. Embedded Security

Embedded Systems haben eine Vielzahl an Architekturen (ARM, MIPS, AVR, 8051, PIC, …); haben oft kein oder nur ein stark spezifiziertes BS; typische eingebettete Systeme haben eine CPU, RAM, ROM (meist Flash) und periphäre Geräte um mit der Außenwelt zu interagieren.

Security ist für eingebette Systeme wichtig, da sie sehr weit verbreitet sind und oft sicherheits-relevante Funktionen erfüllen (z.B. Flugzeugkontrollsysteme) ➔ Attacken können weitreichende Konsequenzen nach sich ziehen

Open vs. Closed Systems: eingebettete Systeme sind (im Vergleich zu PCs) oft closed. Das heißt, die Systeme sind proprietär und es gibt keine Dokumentation zu Interfaces, Protokollen usw.

High-Level-Analyse

Kommunikationsprotokolle werden analysiert ➔ Replay Attacken möglich?; Fuzz Testing; Monitoring der Systeme (crasht das System? , führt es ungewollte Tasks durch?); allerdings kann daraus nicht geschlossen werden ob Implementierung sicher ist

Low-Level Anaylse

Sehr schwierig und zeitraubend; ermöglicht Implementierung zu analysieren (➔ sehr mächtig); beantwortet Fragen wie: ist die Implementierung sicher? Gibt es Sicherheitsmechanismen? Funktionieren diese? Wie weitreichend können Attacken stattfinden?

Ziele von Attacken

Kompletten Zugriff auf Gerätekonsole (root Access bekommen); Software auf Bugs analysieren (z.B. Remote Buffer Overflows finden); gesperrte Features freischalten; alternative Firmware bauen; Geheimnisse extrahieren (z.B. private Schlüssel)

Analysetechniken

Hardwarekomponenten überwachen (z.B. mittels Oszilloskop); Gerätekommunikation überwachen bzw. verändern (z.B: durch Ablöten bestimmter Komponenten); Programmier- oder Debugger-interfaces verwenden

Firmware extrahieren

Firmware kann im internen oder externen Speicher sein; kann z.B. durch herunterladbare Firmwareupgrades gefunden werden; Firmwareanalyse: statisch durch Disassembly, Stringanalyse etc; dynamische mittels Debuggingsetup

Typen eingebetteter Systeme

Kleine Systeme: ohne BS, stark Ressourcengebunden (z.B. Taschenrechner)
(mittel)große Systeme: haben normalerweise BS; Ressourcengebunden (z.B. Smartphone, Router, Modem)

Debugging

Eingebettete Systeme haben oft Debug- und Programmierinterfaces, diese sind aber oft am PCB versteckt.

Side Channel Attacks

Systeme können sicherheitsrelevante Informationen beispielsweise über deren Stromverbrauch verraten. Es gibt viele Side-Channels (Zeit, Stromverbrauch, elektromagnetische oder optische Ausstrahlung, Hitze, ...). Durch Analyse dieser Informationen kann man an Informationen gelangen.

Fault Injection Attacks

Es werden absichtlich Fehler injiziert → Versuch normale Operationen zum Nutzen des Angreifers zu verändern (z.B. Clock Glitching, Voltage Glitching)