Network Security Summary

Andreas Biri, D-ITET 24.01.18

1. Refreshers

1.1 Networking

Protocols & layering divide up network functionality
- each protocol instance only uses services of lower layers
- protocols are horizontal, layers are vertical
- can combine multiple systems with different protocols, layers are transparent / oblivious to higher layers

Encapsulation: use protocol stack & headers
- lower layer wraps higher layer content by adding its own information for delivery, decapsulated at target stack

Segmentation (TCP): divide long messages if larger than MSS (*Maximum Segment Size*) received from application
Fragmentation (IP): divide long message if more than MTU (*Maximum Transportation Unit*) on physical layer

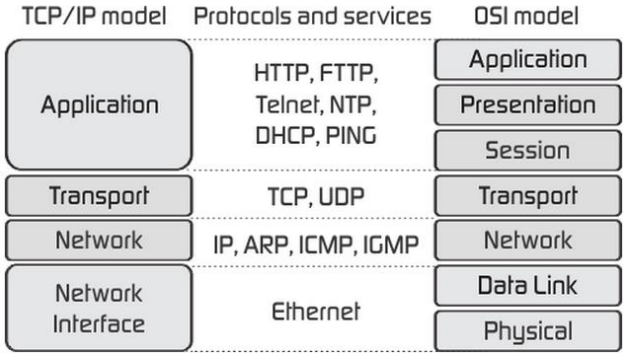
7	Application	– Provides functions needed by users
6	Presentation	– Converts different data representations
5	Session	– Manages task dialogs
4	Transport	– Provides end-to-end delivery
3	Network	– Sends packets over multiple links
2	Data link	– Sends frames of information
1	Physical	– Sends bits as signals

Application	– Programs that use network service
Transport	– Provides end-to-end data delivery
Internet	– Send packets over multiple networks
Link	– Send frames over a link

Internet: "narrow waist", only use IP, many above / below

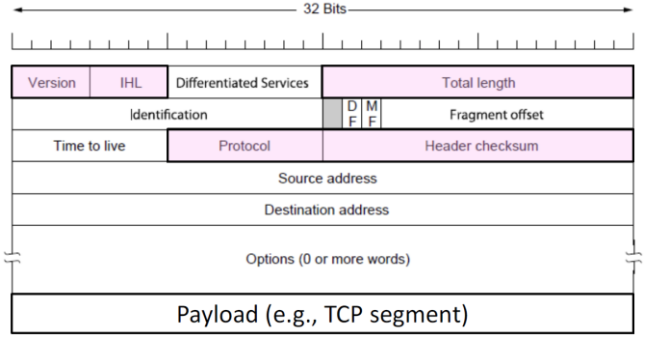
Ethernet: max 1500 bytes (MTU)
- 802.11: has 3 addresses (also AP) & 2300 bytes MTU

Preamble	Destination address	Source address	Type	Data	Pad	Check-sum
8	6	6	2	0-1500	0-46	4



Address Resolution Protocol (ARP): resolving addresses of Local Area Network (LAN) & globally routable ones

IPv4: max size 64k bytes
- together with TCP, get 40 bytes header overhead
- *Traceroute*: use increasing TTL to find all hops on way



ICMP: *Internet Control Message Protocol*
- informs source about erroneous packet
- can contain beginning of violating packet (e.g. too long)

Middleboxes: sit "inside the network", add functionality
- NAT, Firewall, Intrusion Detection System (IDS)
- NAT: has to re-compute checksums (costly)

Switches: extend LAN (Ethernet) on *layer 2*, separate collision domains compared to repeaters/hub

Routers: interconnect networks on *layer 3*
- ARP request don't cross a router (stay inside network)

Routing algorithms

Decentralized, nodes exchange messages with neighbours

Dijkstra: estimate fastest path to destinations iteratively

Link-State routing: learn entire topology by exchanging link lists & then search for optimal forwarding (Dijkstra)
- can be easily secured using digital signatures
- doesn't scale well for large networks (too much info), but fast calculation (compared to DV)

Distance-Vector routing: just learn the distances to the destination & links you need to use to get there
- low overhead, but converges very slowly (e.g. BGP)
- use Bellman-Ford

DNS: *Domain Name System*
- map human-readable addresses to IPs

1.2 TCP / IP

TCP - Transmission Control Protocol
- connection-oriented, error detection & correction
- sequence numbers are byte numbers, full-duplex

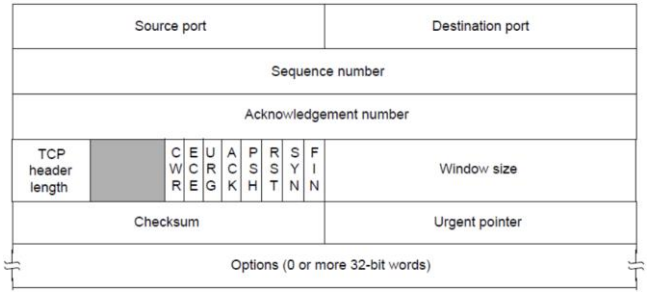
3-way handshake: used both for connection establishment and connection release (using *SYN* & *FIN*)

Sliding window: sender buffers up to W segments until they are acknowledged by the receiver
- slows down sender if too much congestion / overload
- loss detected using timeouts & *retransmit* packets
- adapt round-trip time for current setting
- min of *receiver* window & *congestion* window

Congestion: very low throughput for high loss rates
- if sending above capacity, system will collapse
- allocation should be efficient (all used) & fair (equal)
- "maximize minimal flow"

Additive Increase Multiplicative Decrease (**AIMD**)
- creates "sawtooth" pattern which is efficient & fair
- add α if no congestion, multiply with β if congested
- if timeout recognized, restart with initial window size
- *Slow start:* for each ACK, increase window size by one
- requires only binary feedback from network (ACK/-)
- can be improved using ECN (*Explicit Congestion Notification*)

Fast retransmit: after 3 duplicate ACKs, sender resends packet (even if not yet Timeout reached)



1.3 Cryptography

Secrecy: keep data hidden from unintended receiver
Confidentiality: keep someone else's data secret
Privacy: keep data about a person secret
Anonymity: keep identity of a protocol participant secret

Integrity: ensures data is "correct" (syntax, unchanged)
- locally stored data, should not be corrupted or altered

Authentication: ensure that data originates from sender
- sent over the network, implies integrity of data and correct, claimed sender which created the message

Basic Cryptographic Primitives

Asymmetric: *public-private* key
- much slower than symmetric keys ($\sim 10^{-6}$)
- uses *Public Key Infrastructure* (PKI)
- Diffie-Hellmann key agreement
- used for digital signatures (public to check, private sign)

Symmetric Key: *shared*-key, *same*-key
- Block cipher & stream ciphers
- *Message Authentication Code* (MAC)
- one-way function, cryptographic hash function (no key)

Symmetric Encryption

Encryption Key = Decryption Key

Stream Ciphers: use *pseudo-random generator* (PRG) to generate keystream from seed as a one-time pad
- AES in CTR mode
- vulnerable to Keystream reuse attack, ciphertext mod.

Block Ciphers: pseudo-random permutation (PRP)
- one-to-one mapping of input block to output-block
- encrypt each block separately
- DES, AES (Rijndael)
- block size always 128 bytes, {128,192,256}-bit key
- extremely fast (50 clock cycles for 128-bit key)

Modes of Operation
- ECB: *Electronic code book*
- CBC: *Cipher block chaining*
- CFB: *Cipher feedback*
- OFB: *Output feedback*
- CTR: *Counter mode*
- GCM: *Galois Counter mode*
  (simultaneous encryption & authentication)

Semantic security: adversary knows as much about the plaintext after he saw the ciphertext as before

MAC: provide "cryptographic checksum"
- authentication & integrity
- uses a shared symmetric key
- HMAC-SHA256
- CMAC: Block-cipher based MAC (use last output as MAC)
- only allows authentication, cannot convince anyone else about party of origin (*signatures* can, but slower)

Asymmetric Keys

Diffie-Hellman: use public & private keys
- Public values: large prime p, generator g

$$A \to B: g^a (mod\ p), \quad B \to A: g^b (mod\ p)$$
$$(g^a)^b = g^{ab} (mod\ p) = (g^b)^a$$

RSA: *Rivest, Shamir, Adleman*
- Public key $N = p * q, \quad e$
- Private key p, q, d

Man-in-the-Middle (MitM): asymmetric keys are vulnerable, as you do not know the authenticity of the other party with Diffie-Hellman

EKE: *Encrypted Key Exchange*
- authenticate and establish shared new secret key

Hash function: one-way, weak & strong collision resistance
- *Weak:* cannot find another one for the one you present
- *Strong:* cannot find any two pairs which collide

2. Blockchains

2.1 Bitcoin

Currency needs to be issued by trusted entity (bank)
- users can **withdraw** cash without bank knowing who it is
- merchants can **deposit** received coins back at the bank
- coins are anonymous, even for the bank
- improvements for offline payment & smaller units
- still, required central trusted authority – the bank

→ Currency should be free of any centrally trusted entity

Proof of Work: require work for service requester

Public ledger: prevents double spending
- all actions are publicly known & can be checked

Bitcoin: announced in 2008, first block mined in 2009
- hidden owner makes it attractive to criminals
- hard to precisely model & proof security
- no authority to define specification (take reference impl.)
- 7 transactions/sec, 60min confirmation time (6 blocks)

Consensus is achieved under new constraints:
- no trusted authorities or pre-assumed identities
- assumptions on economic incentives & networking

Components

Transactions: system state presented as set of transactions
- main purpose: transfer money from one entity to another
- cost fees (difference of sums of inputs & outputs)
- transactions are chained & recorded

Output: contains code snipped called *scriptPublicKey*
- contains conditions under which output can be used

Input: contains snipped called *scriptSig*
- typically just complete public Key & signature

Scripting language: stack-based
- script must evaluate to true for transaction to be valid
- *Pay to Public Key Hash* (P2PKH): use public key & signature

Ownership: no explicit user identities or accounts
- public key hash = pseudonym
- knowledge about matching private key allows spending

Mining: prevents double-spending by publishing all transactions in a global, immutable log (*ledger*)
- consist of series of blocks, each containing hash of previous one (*blockchain*)
- need consensus over blockchain, else *forks* appear

Anyone can attempt to extend the chain (append block)
- creating new block is equal to solving computational puzzle called *Proof of Work* (via Brute Forcing)
- average rate is adjustable ($\sim 10min$ on average)
- participants always continue mining on longest chain

Temporary forks: if two valid blocks appear simultaneously
- participants choose randomly on which chain to continue
- system reaches consensus gradually (shorter forks vanish)
- therefore, transaction confirmation not directly, but after approximately 6 blocks have been appended to the chain

Incentives: participants which successfully mine a block are rewarded with a specified amount of Bitcoin currency (*block reward*) to get users to work on the longest chain
- prevents temporary forks from being an issue
- are halved every 4 years until 2140
- *transaction fees* for incentives once rewards are over

As block rewards are significant but very rare, participants collaborate in **mining pools** which leads to centralized computation (which allows for new attacks)

Peer-to-peer network: used to broadcast new transactions
- delay between block discovery & reception should be short to decrease possibility of temporal forks (need to propagate through network in order to be valid)
- if control significant part of the network, allows for denying blocked nodes rewards & transaction inclusion
- choose ~ 8 outgoing connections, ~ 100 incoming
- directory servers (*seed nodes*) help during set-up
- *INV* message (hash) to inform, *GETDATA* to request rest

Analysis

Stability: A system is said to be stable if it provides
- *eventual consensus:* eliminate temporary forks after time
- *exponential convergence:* long forks are very unlikely
- *liveness:* valid transactions will be added to the chain
- *correctness:* in the longest chain, all transactions are valid
- *fairness:* will receive share proportional to work I did

Majority miner (possesses more than half the computation power) could violate all above stability criteria

Selfish miner: temporary withhold multiple discovered blocks in a row & release them after another (shorter) chain has caught up (destroys effort of other users)
- violates fairness if possess 1/3 of processing power

Eclipse attack: abuse connection establishment process
- get victim to connect only to nodes you control
- can isolate node from the rest & use alternative view for double-spending attacks on the victim
- *off-path attack*: does not require compromised network

Message delaying: exploit scalability measure to delay the delivery of victim's packets (high chance of old fork)

Routing-level attacks: assumptions are violated
- efficiency of routing attacks using BGP hijacks
- centralization of Bitcoin miners in small number of ASes
- can perform both network partitioning & delaying msgs

De-anonymization
Tracing flows of money between pseudonyms & mapping them to individuals is often possible

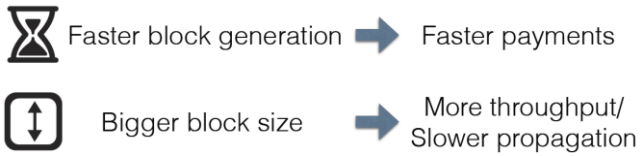
Transaction graph analysis: parse through blockchain & link transactions to identify clusters of addresses
- interact with one address to learn identity (out-of-bands)

Network de-anonymization: use communication directly
- IP addresses are leaked during broadcasts (may use Tor)
- linked pseudonyms use similar set of neighbour nodes

2.2 Other blockchains

Bitcoin has a strong interdependence of block size, generation rate & security
- difficult to upgrade, as need consensus for it
- *hard* fork allows new rules, *soft* fork would be compatible

Bitcoin only allows for monetary transactions
- scripting language limited, could be more expressive



Ethereum: uses a Turing-complete byte-code language
- programming language to express complex contracts
- system can be seen as a distributed virtual machine
- "*smart contracts*" enable financial applications
- create new contracts, invoke functions of existing ones, transfer ether to users & contracts
- contracts can use built-in *pay* function to directly transfer
- sequence of transactions defines state of the contract
- execution fees called *gas*

Validators: can be used to have pre-defined entities to create a permissioned blockchain, not requiring consensus
- can be decentralized, can create our own validators

Hyperledger: does not require "unnecessary" PoW
- can choose either *Proof of Work* (PoW) OR validators
- use *Byzantine fault tolerant consensus protocol* to detect wrong information from the network
- significantly improved performance

Advantages

Decentralization: not a single, centralized trusted entity

Transparency: anyone can verify blockchain validity

3. DNS Security

Domain Name System (DNS): map domain name to IP
- hierarchical structure stored on distributed servers
- responds on TCP/UDP port 53, no authentication
- *Top-level-domains* (TLD): just below the root
- *Second-level-domains* (SLD): companies, institutions
- *Fully Qualified Domain Name* (FQDN): single server

Attack vector:
- easily setup services which are hart to shut-down (Bots)
- helps building hidden channels (tunnelling), as common
- freely available, distributed storage system (text)
- abuse external infrastructure for denial of service
- used for various impersonation attacks
- send small request, receive large response
- use open resolvers to overload victim with responses

Authoritative name server: authoritative for specific zone
- *Resolver:* resolves domain recursively, caches results
- *Root name servers:* 13, controlled by *IANA* (hard-coded)
- should answer only to requests it is authoritative about

(Recursive) Resolver: processes DNS resolution iteratively to provide full answer; *stub* simply forwards request
- *Time to Live* (TTL): tells you how popular server is
- should answer to all requests originating from *its* network

Record types
- *A:* Address record
- *PTR:* Pointer record, used for reverse lookup (IP→Name)
- *NS:* Name server record
- *SOA:* Start of Authority record
- *CNAME:* Alias
- *MX:* Mail Exchanger record
- *TXT:* Text record, can store any text I want

Domain name registrar: manage reservation of SLD names
- includes domain name, holder of domain, NS & addresses

Attacks

DNS Spoofing: respond faster than intended DNS server
- predict TxID and already answer with wrong IP before
- can also spoof NS resolver address during DHCP setup

Cache poisoning: enter wrong data into caches
- locally: change "/etc/hosts/" file with static entries (e.g. point to nowhere for unwanted security addresses)
- *Impersonation attack:* redirect resolver to your IP
- Add multiple (unasked) resolution entries in *Additional* section when your (malicious) address is resolved
- Mitigation: need to match request & response (context)

Distributed Reflection: direct answer to wrong IP
- *Amplification* attack: spoof source IP & request large entry so that intended target receives lots of huge packets
- target gets flooded from legitimate servers (no filtering!)
- Mitigation: source ID verification, only authoritative reply

DNS tunnelling: use for data exfiltration & communication

DNS hijacking: modify registrar entry to point at wrong IP
- can also modify ISP router, or local host DNS settings
- Mitigation: *monitoring services* check correctness of reply

Phantom Domain attack: do not / very slowly respond

Random Subdomain attack: request inexistent subdomain

Botnets: want to centrally control lots of distributed nodes
- should be resistant to hijacking & shut-down attempts
- use layering to protect master *Command & Control* server
- use round-robin IP allocations & very short TTLs
- frequently change IPs by resolving domain to other IP
- can also use multiple different (dynamic) domains
- "rendev-vous" points do not all need to be registered

DNSSEC

Provides origin authentication of DNS data & integrity
- authenticated *Denial of Existence &* signatures
- all data is digitally signed using private key of the zone
- backwards-compatible to normal DNS
- *RRSIG* contains signature, *DNSKEY* the public key

4. Public Key Infrastructure (PKI)

SSL: *Secure Sockets Layer*
TLS: *Transport Layer Security*

Need to make sure that you receive the correct page in the first place (JavaScript encryption can be changed at will)
- server needs to be authenticated to be trusted

Man-in-the-Middle can only be prevented if we authenticate using public keys → **certificates**

TLS session: only server is authenticated

C → S: client_hello
S → C: server_hello
- Ephemeral DH key exchange, RC4 encryption, MD5-based MAC

S → C: Server certificate, containing RSA public key
- Client checks validity + verifies URL matches certificate

S → C: Server_key_exchange: g, p, g^S, $\{H(g, p, g^S)\}_{K_S^{-1}}$
S → C: server_hello_done
C → S: client_key_exchange: g^C
C → S: change_cipher_spec
C → S: finished
S → C: change_cipher_spec
S → C: finished

Public Key Infrastructure (PKI): provides method to validate public keys & distribute them via authentic channels to have *key authentication*

CA: *Certificate Authority*
- *Root CA:* usually the root of trust, mostly offline (can be used to revoke intermediate CAs)
- *Intermediate CA:* actual server responding
- *End-entity certificate:* certificate for a website

Public-key certificate: binds a name to a public key
- signed (indirectly) by the *trust root*
- name usually domain name or email address

Vulnerabilities

Content Delivery Networks (CDN): distributes content instead of original server itself for performance reasons
- requires certificate to service content from domain
- obtain a single certificate for multiple domains

→ **weakest link: i**f one of the domains is compromised, its certificate can be used for multiple others (e.g. in the form of known-plaintext attacks, as all use the same private key)

Compelled certificates: law enforcements possess own intermediate CA certificate, allowing for MitM attacks

Roots of Trusts: do not scale well to the entire world
- Monopoly model: who controls it? DNSSEC, BGPSEC
- Oligarchy model: numerous roots, > 1000 root CAs (single compromised entity enables MitM for all)

Improvements

Free TLS certificates: "Let's Encrypt"
- provide free (short-lived) certificates with automatic domain validation, issuance & renewal
- *Automated Certificate Management Environment* (ACME): put file in specific URL so I can check that you own it

Extended Validation (EV): use multiple certificate levels
- *Organization Validation* (OV): more data in certificate
- *Extended Validation* (EV): more checks before issuing

HTTP Strict Transport Security (HSTS)
- server tells clients that they should only use HTTPS
- prevents downgrading if user visited previously

HTTP Public-Key Pinning (HPKP)
- server tells client a set of public keys to be used

Certificate Revocation List (CRL)
- allow certificate to be revoked (e.g. after disclosure)
- check list before every call to see if revoked (inefficient)

Online Certificate Status Protocol (OCSP)
- verify certificate status, ensure it is valid & not revoked
- response time can be very slow & decrease performance
- *optimistic treatment*: no one sees failure as fatal & aborts
- **OCSP stapling**: server sends OCSP directly in his response (but would also require for intermediate CA, too long)

DNS-based Authentication of Named Entities (DANE)
- authenticate TLS servers without certificate authority
- use DNSSEC to bound certificates to names & choose who client should take to validate the certificate
- heavy reliance on DNSSEC (monopoly model, 1 private key)

Certificate Logs

Ensure that all (also violating) certificates are publicly known; CAs are accountable for their actions (deterrence)

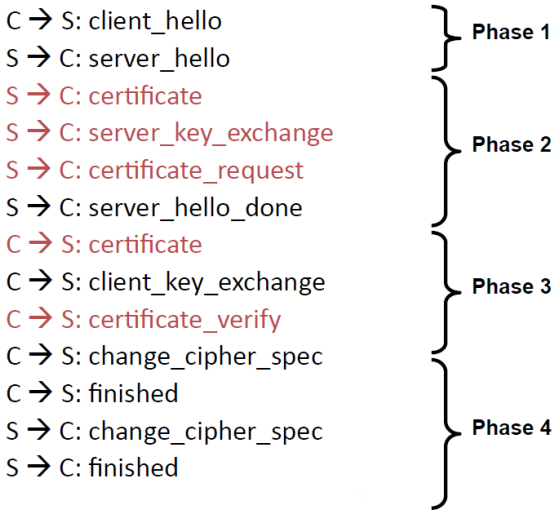
Certificate Transparency (CT)
- make public end-entity TLS certificates public knowledge
- CAs are publicly accountable for all certificates they issue
- append-only list of certificates, periodically updated
- uses *Merkle hash tree* to prevent just being another trusted entity: can check & proof when it misbehaves
- client (*auditor*) & CA (*monitor*) work together to detect
- can simply detect attacks (MitM & false certificates), but cannot prevent them; also no revocations

Signed Certificate Timestamp (SCT)
- "I promise to add it to the log", server sends as proof
- already mandatory for EV certificates

Attack Resilient PKI (ARPKI)
Reduce trust in any single component (CA, log server) by enabling domain to create own security policies
- handle legitimate key & certificate management events
- handle catastrophic events (domain key compromise)
- parties monitor each other for illegitimate behaviour
- use multiple CAs & get signature from all
- compare to Integrity Log Servers (ILS)
- resilience against compromise, need at least n parties

5. TLS

- secrecy to prevent eavesdropper from learning infos
- entity & message authentication to prevent changes



Client is (usually) ned verified, only server is:
Phase 1: "What is supported?" + randoms exchange
Phase 2: Server → Client properties
Phase 3: Client → Server properties
Phase 4: Finalize exchange & double check

Dumbing-down attack: attacker chooses weak versions which client supports to make victim vulnerable

Random Number Generator: best place to attack
- can decrypt everything if I know its state
- undetectable, as looks random to everyone else

Require 4 keys in total:
- 2 for encryption S→C, C→S
- 2 for MACs in both directions

HMAC: use keyed-hash message authentication codes
- MAC not only depends on message, but also key

Key exchange mechanisms

Perfect forward secrecy (PFS)
Compromise of long-term private key does not reveal session key (past traffic cannot be decrypted)

Contributory key agreement
Both parties contribute to the session key, none of them can fully determine the session key (high entropy)
- clients usually not well maintained, old & weak software
- servers might be affected by government, weak by design

RSA: encrypt key with receiver's public key (client choses)
- client picks entire key, no contributory key agreement
- use long-term private key, therefore no PFS
  (but can add additional step with temporal public key)

Fixed Diffie-Hellman: public key certificate contains DH key: $g^S mod\ p$, where S is the long-term private key
- no PFS, but contributory key agreement

Ephemeral Diffie-Hellman: public key signs temp. DH key
- as use temporal keys both on server & client, have PFS
- contributory key agreement, as both contribute to DH

Anonymous Diffie-Hellman: DH without authentication
- PFS & contributory key agreement as before
- not secure against active MitM, as no certificates
- should never be used / always prohibited, as dangerous

In Phase 4, we exchange MACs of the entire conversation
- hash everything so far in the "finished" messages to verify that both sides saw the same messages and an adversary did not temper the exchange
- prevents *Dumbing-down attack* by comparing Phase 1
- send everything once with MD5, once with SHA1 in case one of them should become vulnerable
- MitM still possible if do not use certificates, as simply compare both sides with sides of the attacker

Protocol properties

Key computation
Master secret (MS) consists of the pre-master secret (PS), the client random (CR) & the server random (SR)
- PS is the shared secret obtained during DH
- CR & SR where exchanged in the {client, server}_hello
  (ensure that keys are always different, even with RSA)

$$MS = funct(PS, CR, SR)$$

TLS 1.1 used MD5-SHA-1, whereas TLS 1.2 used SHA-256
TLS 1.1 used concatenation, TLS 1.2 XOR (just like HMAC)

Single-side authentication (MitM attack)
Commonly, client browser doesn't have certificates
- Attacker can easily play the part of the client & open a connection to the server
- However, cannot play part of the server, as he would have to sign his DH key part, which he cant
  (and he doesn't know S, so he cannot simply relay)

Dumbing-down attack
Prevented through the exchange of HMAC at the end to verify that both parties saw the same communication and no attacker tempered with the supported ciphers
- with *Anonymous DH*, MitM is easy & therefore possible

TLS 1.3
Primarily tried to simplify & reduce options, as many allowed attacks on wek crypto functions
- all handshake messages after *ServerHello* are encrypted
- complexity was strongly reduced
- optimized for faster & more efficient traffic
- 1-RTT handshake for naïve clients (can guess capabilities)

C → S: client_hello, g^c
S → C: server_hello, g^s, certificate, signature, finished, application
C → S: finished, application data

- 0-RTT handshake for repeat connections (data in first packet)

6. DDOS Availability

Confidentiality: prevention of information disclosure

Integrity: prevention of modification / deletion

Availability: prevention of withholding of information

Denial of Service (DoS): try to achieve *Resource Starvation*
- *CPU:* excessive CPU load
- *Network / Connectivity:* bandwidth saturation (slow response time result in worse Google ratings)
- *Storage:* excessive memory & storage usage

Distributed DoS (DDoS): large number of diverse compromised systems attack a single target
- fundamental problem: cannot control wo contacts me

Volume-based attacks

Use sheer volume of traffic to limit services

Can either consume bandwidth at the target, or within target and clients (weakest link in the chain)
- ICMP & UDP packet floods
- Reflection & Amplification (e.g. with DNS)

Protocol attacks

Attack IP, UDP, TCP, SSL & reserve resources

Abuse that protocols need to keep server state (e.g. TCP)
- SYN/ACK floods
- TCP connection floods
- Fragmentation attacks (using fragmented packets)

Application Layer attacks

Try to figure out what takes up a lot of resources

Attacks are very difficult to detect, as they require little effort and can have huge consequences (*low-and-slow*)
- SMTP, DNS, FTP, SIP (*layer 7* protocols)
- Application request flow
- Database connection pool exhaustion

6.1 Attack Techniques

Spoofing: hide the origin & redirects traffic to target

Amplification: generate large response with small input
- small request generates large response (DNS)
- single request generates many replies (broadcast ICMP)

Reflection: combine spoofing & amplification to abuse third party infrastructure for powerful, amplified attack

6.2 Attacks

Memory / Storage

Compression bombs: file that unpacks to enormous size
- exceeds available memory of the unpacking server
- reintroduce previously reduced redundancy
- *ZIP*: use nested compression to create gigantic files
- *HTML:* use compressed HTTP response (attack client)
- *Image:* generate gigantic image files (attack target app)
- *Table:* nested tables use a lot of memory

Using bombs, we can attack the decompression engine (applications will hang) and antimalware software & gateways/proxies trying to inspect the traffic
- therefore, should always limit available memory

Session State Exhaustion

Server needs to keep session to continue same conversation by keeping a session state table
- can result in no new connections or even dropped ones

SYN Flood attack: abuse TCP three-way handshake
- simply request too many sessions (client can forget)
- therefore, always try to keep state in traffic / client

Encode the state in a unique but deterministic way that allows the server to validate the state in the reply
- server can re-generate B based on a given function (use salts so that only server can generate it)
- no need to store session state at the server

Slowloris attack: use single machine to take down server
- uses minimal bandwidth & no side-effects on other ports
- keep many connections to target & hold them as long as possible by always adding to the request but never completing it (server will keep connection open)

Therefore, should limit the number of maximal connections from a single source & increase capacity
- impose restrictions on minimal transfer speed & response
- restrict length of time client is allowed to be connected

Furthermore, we can use reverse proxies / firewalls / load balancers to *sanitize the traffic* (make sure limits are imposed & RFCs kept) and distribute the load as we want

Amplification

DNS Reflector attack: abuse external infrastructure
- spoof source address & request large responses
- use UDP as do not require connection setup
- abuse *open resolvers* which answer to anyone

Source IP verification (of ISP) can prevent this
- do not allow traffic from outside to get to you
- no open resolvers which answer requests from anywhere
- only answer to what you are authoritative about

Mail Bounce attack: mail server returns mail if not delivered *for every included recipient* it cannot find
- includes a full copy of the original mail (attachments)

Therefore, do not accept messages if you cannot deliver (stop it already at SMTP level by verifying recipient)
- send at most one error message for an incoming one
- make sure it is always smaller than the original

Smurf attack: send ICMP packets to broadcast address
- all addresses will reply to the spoofed source IP

Never respond and forward packets directed to broadcast addresses which expect a reply

Internet of Things

IoT devices are ideal members of botnets:
- stripped-down operating system is vulnerable
- default (hard-coded) passwords provide easy access
- internet access without any limitations / filtering

Mitigation is even harder than for "normal" systems:
- *Patching* requires robust & easy update mechanism
- *Credentials* should be set by the user upon installation
- *Traffic monitoring* in the network would detect it

6.3 Mitigation

IP Spoofing: prevent address spoofing, identify attacker
- *Ingress Filtering:* validate source IP address field
- *iTrace:* send extra packets to reconstruct attack paths
- *Packet Marking:* routers mark packets to reconstruct IP

Attack prevention: buy DDoS protection & use cloud-based filtering solutions

Attack containment: do not pay & use protection
- even if you want to pay: to whom?!

Recovery procedures
- use immediate failure detection
- use *out-of-band* alerts (no emails about email failures)
- have 24/7 support & existing recovery plans ready

Redundancy: do not rely on single entities to always work
- no single point of failure
- use $N + i$ systems to allow for up to i failures
- use geographically separate locations & connections
- use micro-services which you can gradually scale down
- RAIDs help, but only if you also detect the failure

7. SCION

Internet issues

Routing: sender & receiver have limited control over routing paths, can be easily spoofed using BGP hijacks
Blackholing: advertise more specific IP prefix to gather traffic and simply dump it after analysis

Kill Switches: depend on a few, central entities to work
- communication for geographical area can be halted
- DDoS, BGP hijack, DNS redirection, certificate revocation

Global PKI: all CAs can sign certificates for all websites
- no way to limit trust to CA you decide on
- no single trusted entity anymore, but trust 1000s
- currently security of the weakest link (single *PoF*)

BGP: bad performance of extremely important protocol
- availability can fail for several seconds
- slow convergence with iterative route computation
- susceptible to attacks / misconfigurations
- global results due to local errors (no *fault isolation*)
- poor path predictability, no path choice by endpoints
- bad scalability, as increases linearly with nodes
- only single path, loops can occur upon failures
- no isolation between routing & forwarding (single PoF)

IP / Data Plane
- lack of route transparency (don't know where I go)
- lack of predictability for route availablility
- lack of route choice / control by Tx & Rx
- everyone is trusted and expected to work correctly
- no mechanism to authenticate the source (*spoofing*)

7.1 Architecture

Goals
- high availability, can achieve communication if path exists
- secure entity authentication & transparent operations
- flexible trusts (can actually choose whom to trust)
- path control by both sender & receiver (& can check)
- separation of control (lookup) & data plane (paths)

Control plane

Isolation domain (ISD): each domain has an independent controller & set of name resolution CAs
- can choose your own grouping of ASes
- *ISD core*: *Core ASes* which manage the ISD
- can be in multiple ISDs at the same time

Intra-ISD Path Exploration: Beaconing using PCBs
Inter-ISD Path Exploration: Core Beaconing with ISD cores

Path-segment Construction Beacons (PCB): initiated by Core ISD, PCBs traverse ISD as a (scalable) flood to reach downstream ASes, each representing a path to a core AS
- each AS on path adds AS name, hop field (in & out), MAC
- PCB contain AS signatures & expiration dates (limit time)
- *Up-path:* used from AS to core AS (local path server)
- *Down-path:* used from core AS to AS (core path server)

Each AS possesses both *border routers* & *beacon servers* and can choose which routes it wants to advertise
- **Border routers:** receive the beacons & relay it to a single
- **Beacon server:** coordinate to periodically send beacons to downstream AS (finite number of beacons / time)
- **Path servers**: offer lookup service for path searches

Trust Root Configuration (TRC): per domain, we can decide whom to trust and who can issue certificates which we believe in → Distributed Roots of trust
- new configuration (TRC N+1) must be signed by a threshold number of key from the previous TRC
- network architecture distributes TRC itself

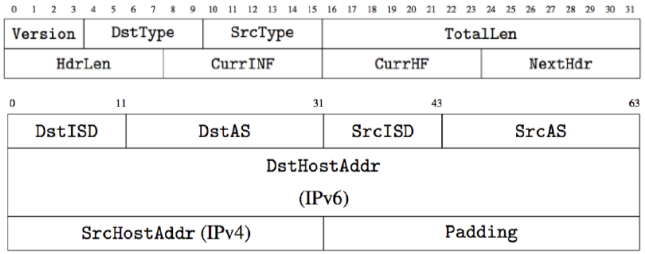
Data plane

Path lookup: *RAINS*, similar to DNS service
- each AS can choose what to advertise to whom
- if not locally cached, local path server will request the down-path segments from the core path servers
- core path server will contact remote core if unknown

1. Host contacts RAINS server with name (Reply: ISD X, AS Y, local address Z)
2. Host contacts local path server for segments (Reply: Up-path, Core-path, Down-path)

Can choose between different variants depending on own preferences (where do I want to rout through)
- *Peering links:* can directly communicate to other AS in other ISD if connected ASes allow such forwarding (harder to censor, as can evade core ASes completely)
- *Shortcut:* if shorter way, don't have to pass core
- can even do *multi-path traffic* (& multi-homing)

Packets: include *address, version, length, headers*
- first *INFO*, then different *hop fields* detailing segments



Included *HMAC* computed using AES with internal, private key as faster & more energy-efficient than TCAM lookup
- simply verify that this is indeed intended egress with MAC, as only AS itself can calculate the MAC, instead of lookup in a table (as traditional routers would do)
- while packets larger, routing becomes faster & efficient
- if route not advertised, attacker has to guess MAC

Ingress & Egress Interface Identifiers: each AS assigns unique integer to each interface to neighbouring ASes
- use internal routing protocol to get from ingress to egress

7.2 Public Key Infrastructure

Control plane: system to determine & disseminate paths
- currently for inter-domain paths: BGP + ICMP
- need high availability, cannot require routing for it

ISD: subset of Internet can agree on roots of trust
- authenticate entities within ISD
- each AS chooses its own ISD (and therefore root of trust)

TRC: contains trust roots for three PKIs
- *control-plane* PKI: core AS certificates
- *end-entity* PKI: root CA and log server certificates
- *name-resolution* PKI: root name server certificate
- neighbouring TRCs sign each other so that they can be changed if a threshold value agrees to the changes
- TRC verification of other ISDs follows core paths
- also contains threshold for number of signatures for SCP (minimal number of trusted entities which would need to be malicious in order to forge an SCP & MSC)

Control-plane PKI

AS certificates are signed by a core AS (as defined by TRC)
- only short lived so revocation is not necessary
- core AS certificate can be revoked through TRC update
- certificate distribution is tied to path exploration
- main focus is on availability, not security

End-entity PKI

Subject certificate policy (SCP): policy which all of a domain's certificates need to adhere to
- list of trusted CAs (which the *domain* trusts)
- threshold for number of signatures required for MSC (minimal number of CAs which would need to collude)
- might require *proof of absence* if none is existent in the TRC you currently use (use MHT & show not where should be)

Multi-signature certificate (MSC): domain certificate signed by multiple entities & signed by SCP
- can be received from multiple CAs

Name-resolution PKI

Double verification path: use two independent sources
- all delegations in name resolution are signed (name-resolution PKI for high *availability*, get new one)
- domain entry itself is also signed by SCP (SCP guarantees high *security*, as can choose root of trust)

7.3 SCION in detail

Secure control plane messages

Each AS has a certificate: $\{AS, K_{AS}, expiration\}_{K_{Core\ AS}}$
- each PCB is signed by core AS when flooding starts
- each AS which resends PCB signs the updated variant

Failed Interface Detection: periodic keep-alive messages
- if absent after some time, link is declared inactive

Secure Path revocation: AS adds *Revocation Token* to PCB
- if cannot be forwarded, distribute token to all concerned

Anycast: send to anycast address instead of destination
- border routers will determine whom to send to

SCION Control Message Protocol (SCMP)
- equivalent to *Internet Control Message Protocol* (ICMP)
- can use asymmetric (AS certificates) & symmetric (DRKey)

DRKey

Rapid establishment of shared secret key for performance
- enables per-packet source authentication with shared key

Dynamically Recreatable Keys (DRKey): use per-AS secret value to derive key with *Pseudo-Random function* (PRF)
- use secret value known only to entities inside same AS

0^{th} order: per-AS local secret key (updated frequently)
1^{st} order: key establishment towards other ASes
2^{nd} order: locally derived symmetric keys for end hosts

Certificate servers inside each AS provide key exchange, local key establishment & key management
- get shared symmetric key without costly key exchange between communicating parties (only local server)
- keys are prefetched between certificate servers (use AS certificates) and can be directly forwarded to end-entity
- create second-order key for AS-to-Host or even H-to-H:

$K_{X \to Y} = PRF_{SV_x}(\text{"Y"})$: key from AS X to AS Y
$K_{X \to Y:H} = PRF_{K_{X \to Y}}(\text{"}H\text{"})$: key from AS X to host H in AS Y

Routers inside X can create all keys (if they know SV_x) and therefore enable per-host authenticated error messages
- key derivation can be delegated to entities without even contacting the certificate server anymore

Can even use protocol-specific keys using appropriate arguments for the PRF, based on the per-host key
- e.g. all DNS servers of Y receive $K_{Y \to X} = PRF_{SV_x}(\text{"}Y\text{"})$ and can then derive a second-order key to answer requests from any host H in AS X with $K_{Y \to X:H} = PRF_{K_{Y \to X}}(\text{"}H\text{"})$

Allows efficient **computation of MACs** even for first packet
- very fast verification of authenticated messages and black-listing of malicious sources if detected

ISD coordination

Globally decentralized system to add new ISDs
- no one should exclusively control ISD membership
- no coalition should be able to exclude from joining
- want to provide maximal transparency

Use PCB extensions to announce through neighbours
- each ISD has a globally unique identifier
- keep in quarantine for 7 days & resolve conflicts
- blacklists will prevent false entries from being accepted

8. Anonymous communication

IP addresses leak *metadata*: who talks to whom, how long
- may want to hide from the destination itself (retaliation)
- TLS only encrypts the content, doesn't hide traffic

Anonymity
Sender: adversary knows receiver, may learn message
- if the sender anonymity is small, only little anonymity
Receiver: adversary knows sender, may choose message
- might exist a pseudonym for the receiver (hidden service)
- *Third party anonymity*: trust dest, but not network

Unlinkability: while adversary knows both sender & receiver, the link between the two is unknown
- requires multiple users to talk simultaneously
- Anonymity → Unlinkability

Unobservability: adversary can't tell whether communication is even taking place
- use DSSS or simply always keep sending traffic
- Unobservability → Anonymity

8.1 Mix-nets

Batching: collect a number of messages before forwarding

Mixing: change the order of the messages before sending

Intersection attack: see destination set of messages
- can do statistical analysis to detect who is talking

Cover traffic: prevent statistical disclosure

Ask a *mix* whether he has stored packets for you
- if yes, receive valid message for you as destination
- if no, receive dummy message so that not seen

Mix cascades: use multiple mixes to avoid single point of failure, as might not trust all mixes on the path
- sender encrypts return address with **layered keys** (every mix decrypts his layer & forwards accordingly)

8.2 Circuit-based systems (Onion routing)

Mix-nets only offered low performance and large delays
- reduce anonymity by only using **layered encryption**
- use a **virtual circuit** which remains for all packets
- use *symmetric key crypto* for large speed gains
- only assume *local* adversary (sits just at points)

Each relay receives a key & a tag of the next relay
- every link in-between pairs is (also) encrypted

Circuit setup: sender negotiates shared keys with relays
- requires asymmetric crypto; relays store keys
Data forwarding: packets are forwarded along the circuit
- use only symmetric key crypto (AES)
Circuit tear-down: release state on relays against attacks
- circuits have a limited time-to-live of 10min (ToR)

Setup methods

Direct circuit setup: establish state on relays by using a normal packet as for mixes (send to all linearly)
- fast, sender knows all established keys in advance
- if relay keys are disclosed, can all be decrypted

Telescopic circuit setup: keys are negotiated one at a time
- circuit is extended by one hop and then reply goes back
- much slower, as want to have geographically distributed
- offers *forward secrecy* (unlike direct circuit), as we create temporal keys between all entities
- circuits are teared-down in opposite order (back-to-front)

Attacks

Traffic analysis attack: flow & website fingerprinting
- add watermark / fingerprint and try to detect at the end
- try to find traffic pattern & compare to known websites
- use cover traffic & mixing to prevent (adds overhead)

Confirmation attack: change at entry, observe at outcome

Higher layer attacks: try to probe stack & find hints
- use per-hop TCP to prevent TCP stack fingerprinting
- use ToR browser so everyone seems same at HTTP layer

8.3 ToR

2 million users daily, more than 6000 relays

Entry guard: input into the ToR network

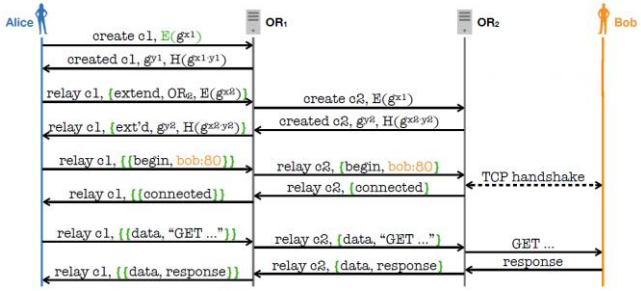
Middle relay: one out of 3 relays in the circuit

Exit relay: only visible point for outside viewer
- is legally responsible for its traffic (unwanted)
- can have *exit policies* to limit destinations they deliver

ToR uses many of the conceps of mix-nets (*Chaum*)
- telescopic setup (for PFS)
- per-hop TCP to prevent stack fingerprinting
- per-hop TLS except for the last hop
- ToR browser cleans HTTP/HTTPS traffic
- end-to-end integrity checks (client → exit relay)

Cell (512 bytes) is the basic unit
- contais *circuitID* & *command* fields
  (re-written at every relay, use relay states and circID to map incoming ports to outgoing ports)

Digest (hash of decrypted content) is checked:
- if correct, check command
- if not, replace circuitID and forward cell along



To prevent arbitrary long loop (DoS on ToR infrastructure):
- *create* cells can only be contained in *relay_early* cells
- each relay only allows 8 *relay_early* per circuit
- therefore, maximal length capped at 8

Hidden services

Hash of public key is identifier for its hidden service

Introduction points (IP): allow for setting up *rendezvous*
- delegation reduces load on IP, as can meet up anywhere

Rendezvous: both parties connect to rendezvous & hand over packets there, not observable in rest of the network

Directory services

10 *directory authorities* (servers) track state of relays & store their public keys, run consensus algorithm
- clients can update lists and prevent attacker from giving it
- DAs verify that relays broadcast correct information
- Limit number of relays from same IP subnet
- Relays act as *directory caches* for scalability

Bridge relays

Not publicly listed relays (distributed through friends) to evade censorships which simply black-list ToR relays
- also, use *pluggable transports* to hide ToR traffic
- always need to be updated & renewed (arms race)

8.4 Decoy routing

Use friendly ISPs to circumvent censorship
- special routers extract information from covert channels
  (e.g. TCP initial sequence number only seems random, but can transmit 24 bits per connection setup)
- establish secure connection to router so that traffic is rerouted from decoy destination to real destination
- requires state on router & asymmetric crypto (expensive)

First, initiated needs to know the router's public key
- then, starts DH to create a secure channel
- router uses shared key to deflect connection
- router acts as proxy & substitutes addresses in both directions so that censor does not realise that only decoy

8.5 Network-layer anonymous security

Lightweight Anonymity & Privacy (LAP)

Autonomous system (AS) takes over role of relay
- uses symmetric crypto (instead of asymmetric one)
- no detours by taking same path as normal traffic
- only limited router state (keep packet-carried state)
- requires full trust in your own ISP

Provides anonymity in case of compromised destination
- good anonymity for very small performance costs
- each AS encrypts & MACs their per-hop routing info
  (only it can decrypt again on the way back)
- use same header for both directions, index shows pos.
- increased anonymity set with more AS in-between
- no payload encryption, can see how many hops in header

APNA

ISP sometimes require accountability (e.g. for government)
- act both as *privacy broker* & *accountability delegate*
- only sender & destination ISP are involved (trusted party)

ISP creates strong notion of identity *within* ISP boundaries
- every packet can be attributed to the sender by its ISP
- hides host identity *outside* of ISP (only AS number leaked)
- uses *ephemeral IDs* (temporal) for its customers

Hosts obtain EphID & certificate of other party
- then, use private key to exchange shared secret key
- even ISPs cannot decrypt the data (*data privacy*)
- key between AS & host used to create MAC (*Src auth.*)

HORNET

Full onion routing (allows stronger thread model)
- tunnel setup with asymmetric crypto
- *layered payload encryption*, requires path-aware routing
- *stateless forwarding* (routing info in headers)
  (but try to hide path length & current position on path)
- allows for asymmetric paths (not same in both directions)

9. Firewalls & Intrusion detection

9.1 Firewalls

Protects & separates trusted network from untrusted one
- allows all authorized communications to pass
- access control policy between networks

Network firewall: filter traffic between networks
- protects different network segments (machines)

Host firewall: layer of software on single host

Ingress: traffic from *un*trusted network to trusted one
Egress: traffic from trusted network to *un*trusted one

Rules match 3 action for each packet structure:
- *accept:* packet can pass
- *drop:* access is denied, no one is informed
- *reject:* access is denied, inform source about it (ICMP)
- can have *default reject* (against unwanted behaviour) or *default accept* (for minimal interference with traffic)

Firewall types

Stateless: decision based on packet header information
- use network layer information (IP, port, flags)
- good performance, application independent

Stateful: keep track of the state of network connections
- decision based on session state (requires saving it)
- state can be inconsistent, can be exploited for DoS
- *NAT:* everything from outside is blocked if not requested

Next Generation Firewall (NGFW)
Perform *deep packet inspection* of all ports & protocols
- compare packet against protocol specifications
- need to support many protocols & work efficiently

Web Application Firewall (WAF)
Protects web-based applications from malicious requests
- reverse proxy outside of internal network, checks patterns & lists to prevent e.g. SQL injection

9.2 Intrusion detection

While firewalls are useful, they are limited in detection:
- encrypted traffic cannot be inspected
- high number of false positives
- limited processing power & latency for high-speed links
- application level attacks cannot be detected (JavaScript)

High number of false positives costs lot of time to inspect, increases the latency of the inspection
- often not given, but as important as *true positives* rate
- *false negatives* (unprotected) central, but hard to learn

Detection

Reactive: system only detects known, fingerprinted attacks
Proactive: system can detect known & unknown attacks
Deterministic: always performs given same input (e.g. lists)
Non deterministic: fuzzy logic, depends on current state

Protocol analysis
- analysis & decoding of protocols, traffic reassembly

Signature-based detection
- *Blacklist/whitelist:* compare attributes of observed traffic (One-dimensional: simply compare to known list)
- *Pattern matching:* compare attributes to known malware (Two-dimensional: compare expressions & strings, flexible Multi-dimensional: add weights of suspicion & threshold)
- signatures created manually & regularly updated
- only reactive (needs to be known), but very fast

Sandboxing
- suspicious file is executed in a virtual environment
- proactive, can detect unknown threads by analysis
- expensive to execute, high latency

Machine learning
- recognize complex patterns based on previous data
- *supervised:* using known set, train model to detect
- *unsupervised:* find hidden structure in unlabelled data
- can detect new attacks by analysing traffic

Attacks

IP Source spoofing: bypass filter by giving wrong address
- does not work on TCP traffic, as requires handshake

Artificial fragmentation: split packet into multiple part
- looks different on endpoint/host than on the network
- can do reordering, overlaps, introduce redundancy

Vulnerabilities: exploit firewall software / OS / target app

Denial of Service: state explosion, use fallback policies
- "if I know what you reject, can even lock you out!"

Tunnelling / Covert channels: exfil data from network
- use data in ICMP packets, DNS requests

Encodings: encode & add noise to hide traffic
- transform into structure which is not seen as bad
- millions of combinations to hide traffic
- require exact same implementation everywhere to detect

9.3 Malware Development & Evasion

Usual detection is based on two approaches
- *Static (signatures):* identify same, known structure
- *Dynamic (behaviour):* identify suspicious acting

Malware development life cycle

1. Develop new malware of desired functionality
2. Create array of unique samples of initial malware
3. Protect samples from analysis ("encrypt")
4. Make samples aware of detection (sandboxing)
5. Quality assurance: test already against anti-virus

Serial variants / permutations: automatically churn out new variants of same malware on massive scale
- only release the ones which are not yet detected
- release them in waves so that can continue attack while old ones are detected by malware detection systems

Hardening: use commercial software protection tools

Crypters: encrypt malware so that signature detection systems & static analysis tools are ineffective
- only decrypt sections of code which is being executed

Protectors: add anti-debugging features so that samples cannot be dissected by researches & sandbox analysis
- detect VMs & Debugging mode by analysing environment
- originally developed as DRM protection technology

Packers: make binaries smaller & more portable
- speed up infection, more difficult to detect
- can unpack differently every time to change structure

Polymorphism techniques: mutate code without changing
- reorder & replace, insert noise code which never execute
- use different compilers to change binary code

Binders: pack malware onto valid .exe (embed malware)

9.4 Layered Security

Direct attack: initiated by attacker
- exploit services which are always reachable
- "server-side" exploits against vulnerable app / OS

Indirect attack: initiated by target
- attacker has little influence on timing of thread
- uses infected system to make itself vulnerable

Unfortunately, simply linearly coupling multiple defence systems doesn't help much, as failure rate is often strongly correlated & exploits not discovered by multiple IPS

Therefore, should always plan of how to handle a successful breach in advance
- deploy tools to quickly detect & remediate attacks

Zero-day exploit: take advantage of vulnerability which is not known (& patched) before

10. Broadcast Authentication

Sender uses broadcast channel to disseminate data
- receivers have unicast communication to sender
- broadcast sender is lossy (use FEC)

Forward error correction (FEC): reconstruct packets

Authentication: symmetric cryptography (shared key)
- receiver can only convince herself that data was generated by the sender (as could simulate itself)

Signature: asymmetric cryptography (PKI)
- receiver can prove to third party that data was generated by sender (as only it knows the corresponding private key)
- much more expensive to create (10^5)
- *non-repudiation:* sender can't deny that it sent message

MAC systems

Single MAC broadcast authentication
Sender attaches single MAC to each packet
- every receiver obtains MAC key K to verify
- very efficient, secure against external adversary
- not secure if a single receiver is malicions
- no non-repudiation, as all receivers could create one

Multi MAC schemes
Each member only gets a subset of (many) keys
- each message is authenticated with all MACs
- receiver checks message with all keys it knows
- large overhead, not secure for many colluding receivers

Authentication needs asymmetry

- only sender should be able to generate a new message
- receivers should only be able to verify its correctness
- signatures are extremely expensive to generate & verify
- susceptible to signature flooding attack

Signature flooding: overload receiver with fake signatures
- receiver is busy checking wrong data and fails to work

Asymmetric cryptography

Use short-lived small RSA keys (384 bits)
- periodically send out new public key with strong sign.
- relatively low computation overhead, no delay
- time synchronization required, not robust to packet loss
- does not provide non-repudiation, no long-term security

TESLA: *Timed Efficient Stream Loss-tolerant Authentication*

Use only symmetric cryptography for good performance
- asymmetry via *delayed key disclosure* (requires time sync)

Receiver knows key disclosure schedule and will be able to check correctness of MACs in retrospective (short delay)
- need to know that during transmission, sender did not yet disclose K (must still be secret)
- upperbound maximal time synchronization error
- digital signature for initial authentication
- key disclosed 2 time intervals after use

One-way hash chains: calculate chain of hashes (keys) & release them in reverse order of construction
- infeasible for anyone to predict next value
- efficiently authenticate all values using & verify chain
- add index ($r_i = H(r_{i+1}||i+1)$) to make it even more difficult to forge it (longer chain would be easier)
- can tolerate packet loss, as can jump single steps

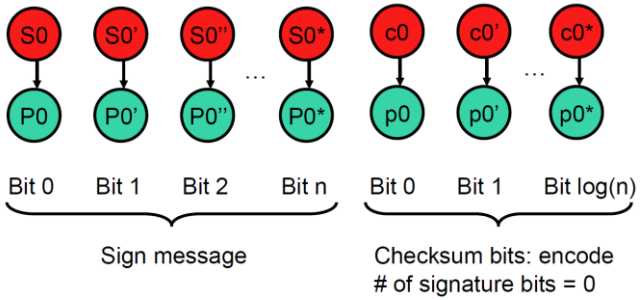
Disclosed value of key chain is a public key, it allows authentication of subsequent messages
- receivers can only verify, not generate packets
- with trusted time stamping, TESLA can provide signature properties with only MAC overhead

Requires storage of packets until have been verified (therefore vulnerable to storage-based attacks)
- delayed authentication
- requires time synchronization

One-Time Signatures

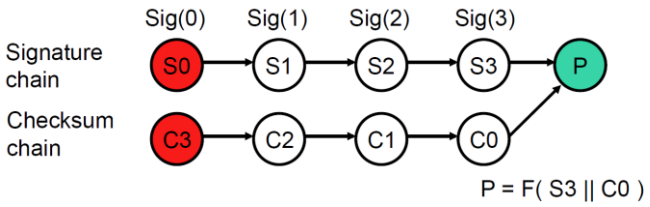
Standard signatures are expensive to generate & verify
- amortise single signature to sign multiple messages

Cryptographic hash function: map arbitrary length input to finite length output (SHA256, SHA-3)
- *one-way*: cannot reverse function & find input
- *weak* collision resistance: cannot find same hash again
- *strong* collision resistance: cannot find pair of hashes



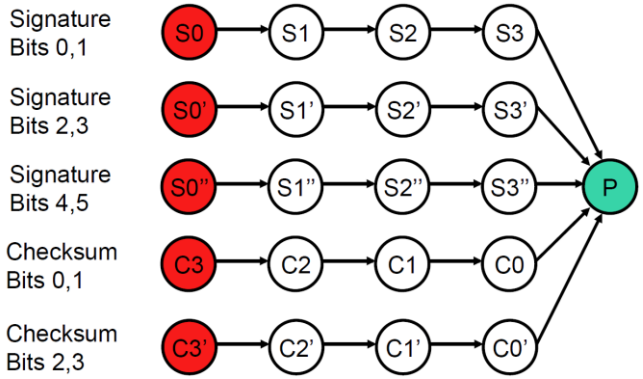
First, hash the message to get fixed 128 bits
- then want to sign this hash, which only I can create
- for every single bit of the hash, disclose either "1" or "0"

To transmit "0"-bit, send S0; to transmit "1"-bit, send P0
- last bits give checksum over numbers of "0"-bits
- can only increase number of 1s by applying hash function
- to forge message, need to invert ≥ 1 one-way hash fct
- *non-repudiation*, as no one else could forge it



Checksum chain which goes to the other way
- if I want to send "0", just send "S0||C0"

Merkle-Winternitz construction: encode sum in checksum
- again, checksum chain goes the other direction (last 2)
- only need to publicly share (digitally sign) *P*



For low-entropy data, can even directly encode message like this (e.g. use 6 message bits) & prove it directly
- offers *non-repudiation*, as need to know private keys
- very efficient, as only require single PKI usage

Stream signatures

Receiver gets sequence of packets & wants to authenticate
- cannot handle packet loss, as then hash / key lost

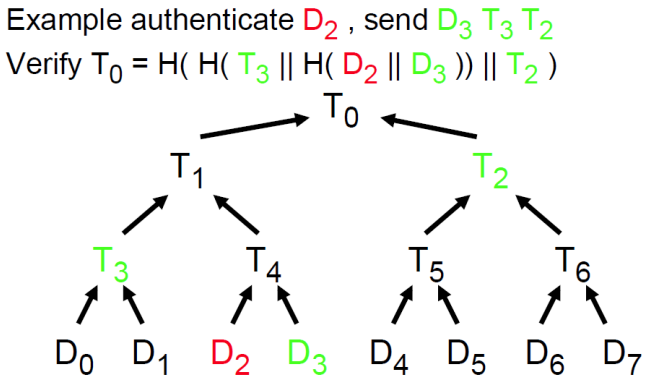
Chained hashes: always include hash of next in previous packet, first hash is authenticated using PKI
- gives you *non-repudiation*, as no one can change msg

Offline: sender knows entire stream before sending
- each packet authenticates the next one, first with PKI

Online: use Merkel-Winternitz construction
- one-time public key use for each packet (only once)
- bootstrap next packet with key from previous one
- first packet is signed using private-key

Merkle Hash Trees (MHT)

Allows for efficient verification of belonging to group
- build a tree where each packet corresponds to one leaf
- sign root (T_0) with private key & send to verifier together with all other required hashes
- verifier can then efficiently check for each packet that it belongs to the tree, and as root is authenticated, packet is
- high buffer overhead at sender, as needs to store tree



Signature Flooding attacks

Overwhelming receiver with signatures works well with traditional methods, as very expensive to verify

TESLA: offers non-repudiation with signature and small costs to verify, as all symmetric crypto

Merkle Hash Tree: binds signature together with data
- only verify if attacker built correct MHT, therefore forcing attacker to send more & less signatures to verify packets

Vehicular Ad Hoc Network (VANET)

Use IEEE 1609.2 as a standard for secure communication

Elliptic Curve Digital Signature Algorithm (ECDSA)
every message contains a signature for non-repudiation
- can prove to third party that something malicious happened & support efficient multi-hop authentication

11. Probability monitoring

Inspecting packets to measure the amount & type of traffic for better network management & anomaly detection
- accounting (usage-based pricing) & traffic engineering
- volume-based attacks (DoS) or port scans
- gather statistics using only limited resources
- analyses packet headers only, can find anomalies

200'000 concurrent flows on 1 Gbps link
- global IP traffic: 10^{15} bytes / month
- average packet size: ~ 270 bytes

Flow identifier (FID) is usually a 5-tuple:
- Src IP & Dst IP
- Src Port & Dst Port
- Protocol

Probabilistic Traffic monitoring

Trade accuracy for efficiency
- only estimate traffic instead of precise data
- summarize all traffic into compact dataset

General purpose measurement

NetFlow: perform packet instead of byte sampling
- requires less writing & memory lookups
- can over- & under-estimate, bad for billing purposes

Packet sampling: sample one every k packets to estimate traffic patterns of flows over entire time span
- bias towards small packets, as probability indep. of size
- requires state for every single flow detected
- sampling with constant period can hide periodic traffic (might not be representative for general situation)

Gathering information for all flows is inaccurate, need to store specific information about them:
- large flows & subpopulations (e.g. small-volume flows)
- number of flows & flow distribution
- address access patterns

Identifying large flows

Elephant flows (large flows) consume largest amount of bandwidth and therefore require special treatment
- classified as elephant if requiring more than threshold of link capacity during given measurement interval
- requires keeping much less state while still getting infos

Sample-and-hold: sample each byte with probability p
- large packets are much more likely to be seen, as for packet of size s, probability is $1-(1-p)^s \approx p*s$
- *hold:* if sampled once, always update flow entry (requires lookup for every incoming packet)
- if sizes larger than threshold, keep as elephants
- no overcounting (good for billing, as lower-bound)

Multistage filter

Stage: hash flow ID & increase corresponding counter
- counter accumulates *number of bytes* for each flow
- each stage uses an independent hash function
- flow is considered *large* if sum of counters $\geq$ threshold
- no *false negatives* (capture all large flows)
- *false positives* decrease exponentially with nr of stages
- can use crypto functions (AES) for very fast "hashing"

Majority algorithm

Exact, linear-time algo to find frequent items in 2 passes
- requires $1/\theta$ space for finding flows larger than fraction θ

1. Identify all frequent items (no FN) / candidates
   (If counter = 0, store current item
   If not, increase if same, decrease all if different)
2. Eliminate all FP by dropping too low ones
   (If not enough, then last ones are random)

EARDET

Find all items that appear in stream of size m more than k
- limited space of $n = m/k - 1$ counters
- if no counter free, decrement all by 1; else, fill in
- use packet size instead of number of items for precision
- decrement regularly for long-lived flows
- blacklist identified large flows over given threshold
- no FN for large flows, no FP for small flows

Finding duplicates

Bloom filter: provides probabilistic structure for membership test without storing all previous elements
- m : number of bits in BF
- k : number of hash functions

1. Use k hash functions to map to $0 \leq h_i < m$
2. For all h_i, set corresponding entry to "1"
3. When testing and one not "1", no duplicate

Never *false negatives*, but might be *false positives*
- filter does fill up over time, therefore more FPs
- require resetting them from time to time
- use 2 Bloom filters & rotate for increased duration
- again, use crypto functions for very fast "hashing"
- O(1) membership checking

Estimating number of flows

Use Bloom filter to test whether new flow
- if new flow (no duplicate), increase counter
- fast membership check, but still $O(N)$ memory overhead

Probabilistic counting: hash flow ID to generate $0 \leq n \leq 1$
- only keep flow associated with smallest hash value
- more flows → higher probability of small hash
- *assumption:* hash distribute value uniformly over interval

$$\#\ of\ flows\ \approx \frac{1}{n_{min}}$$

Relying only on the single smallest value is not robust
- *min* has a higher variance than *median*
- single attacker can bias the estimation

→ use k −th smallest value is more robust

$$\#\ of\ flows\ \approx \frac{k}{n_{\text{kth min}}}$$

Framing attack: cause innocent flow to be punished

12. Course Summary

$2^n = 10^m$: $m \approx (n/10) * 3 \approx n / 3.3$

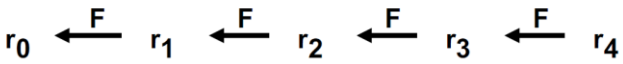
$n \approx (m/3) * 10 \approx 3.3 * m$

Seconds per day: 2^{16}

Seconds per year: 2^{25}

One-way Hash Chain

- **Pick random r_N and public one-way function F**
- **$r_i = F(r_{i+1})$**
- **Secret value: r_N , public value r_0**

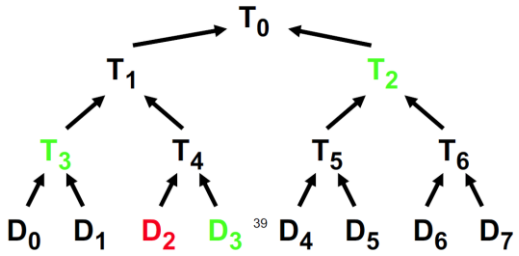


- use in reverse order, not possible to derive next one
- robust to missing values & efficient authentication

Merkle Hash Tree (MHT)

Example authenticate D_2 , send D_3 T_3 T_2

Verify T_0 = H(H(T_3 || H(D_2 || D_3)) || T_2)



- Verifier knows T_0 (signed by trusted entity)
- can now very fast authenticate that D_2 is in the tree:
  give D_3, T_3, T_2 & D_2 → verify that it will result in T_0

CAP: *Consistency, Availability, Partition tolerance*

- (most recent)(always response) (allow packet drops)
- impossible to achieve all 3, need to sacrifice one

Coremelt attack: congest network, not target

- route thousands of small links over same node so that the traffic to the destination is disturbed without realising