Network Security: Protocol Analysis, Firewalls

Tuomas Aura, Microsoft Research, UK

Outline

- Tools for protocol analysis
- Stateless packet filter
- Dynamic packet filter
- Transport and application-layer firewalls
- Firewall issues

Tools for protocol analysis

Dolev-Yao model

- Standard model for protocol analysis:
  - Network is the attacker. Honest nodes send messages to and receive messages from it
- Honest nodes are simple reactive state machines:
  - Honest nodes initiate protocol runs, respond to received msgs, and move from state to state
  - Honest nodes can execute multiple protocol runs in parallel: unlimited parallel instances of the state machine
  - Honest nodes do not reveal their secrets
- Attacker plays a game:
  - Attacker has some initial knowledge, such as public data, the secrets of several corrupt nodes, and old session keys
  - Attacker can receive initial messages from honest participants
  - Attacker's knowledge grows from each received message
  - Attacker can perform message decomposition, composition, and cryptographic operations —for encrypted and authenticated messages only if it know the necessary keys
  - Attacker can send messages to honest participants to trigger response and state change
- Attacker tries to reach an unsafe state where some security goal is broken
  - Unsafe states defined based on attacker's knowledge and honest nodes' state, e.g. A thinks it shares K_{ses} with B, and the attacker know K_{ses}

Protocol analysis tools

- Examples of tools for verification of cryptographic protocols:

  Proverif — Bruno Blanchet (ENS, Paris)
  - Horn clauses and Applied Pi calculus

  Constraint Solver — John Millen (SRI) and Ricardo Corin (INRIA-MSR)
  - Prolog, CAPSL language for protocol specification

  Isabelle theorem prover — Larry Paulson (Cambridge)
  - Lambda calculus, high-order logic (HOL)

  Murphi — Stanford/Utah
- Practically every formal analysis method has been tried

Proverif example

- Demo of protocol modelling with Proverif
  - Proverif models are usually written in Pi calculus, which is relatively easy for a programmer to write
  - We use Horn clauses — closer to the Dolev-Yao model
- Recall the Needham-Schroeder public-key protocol:
  1. A → B: $E_B(N_A, A)$
  2. B → A: $E_A(N_A, N_B)$
  3. A → B: $E_B(N_B)$

  N_A, N_B = secret nonces

  $K_{ses} = h(N_A, N_B)$
- Dolev-Yao model:
  - Network = attacker
  - Honest parties are simple reactive state machines
  - Attacker's knowledge set grows over time
- Security proof in Proverif is sound
  - Sound security proof = complete algorithm for finding all attacks
  - Approximations → may sometimes find attacks that are not real

Function and symbol definitions

- Public-key encryption:
  fun encrypt/3. encrypt(tag, msg, pk) = E_{pk}(msg)
- Nonces:
  fun Na/3. Na(x,y,s) = x's nonce for y in the role of A
  fun Nb/3. Nb(x,y,t) = y's nonce for x in the role of B
  fun Nc/1. Nc(u) = Attacker's nonce
- Type tags for messages (could also leave out and maybe find more problems):
  fun Msg1/0.
  fun Msg2/0.
  fun Msg3/0.
- Host and their states:
  fun H/1. Honest host H(i)
  fun C/1. Corrupt host C(i)
  fun stateA/3. stateA(A,B,N_A) = state in role A after sending Msg 1
  fun stateB/4. stateB(A,B,N_A,N_B) = state in role B after sending Msg 2
  fun acceptKeyA/4. acceptKeyA(A,B,N_A,N_B) = final state in role A
  fun acceptKeyB/4. acceptKeyB(A,B,N_A,N_B) = final state in role B

7

Attacker's capabilities

- Create principals, either honest or corrupt:
  c:H(i);
  c:C(i);
- Attacker knows the message tags:
  c:Msg1;
  c:Msg2;
  c:Msg3;
- Attacker may generate new nonces (not actually needed):
  c:Nc(u);
- Attacker's cryptographic computation:
  c:encrypt(tag, x, C(i)) -> c:x;
  c:tag & c:x & c:h -> c:encrypt(tag, x, h);
  c:tag & c:x & c:y & c:h -> c:encrypt(tag, (x,y), h);
- Note: "c:" is a predicate meaning "attacker knows"

8

Honest principals' behavior

- Role A:
  c:H(i) → c:(stateA(H(i),b,Na(H(i),b,s)),
  encrypt(Msg1, (Na(H(i),b,s),H(i)), b));
  c:stateA(H(i),b,na) & c:encrypt(Msg2, (na,nb), H(i))
  → c:(acceptKeyA(H(i),b,na,nb),
  encrypt(Msg3, nb, b));
- Role B:
  c:encrypt(Msg1, (na,a), H(j))
  → c:(stateB(a,H(j),na,Nb(a,H(j),t)),
  encrypt(Msg2, (na, Nb(a,H(j),t)), a));
  c:stateB(a,H(j),na,nb) & c:encrypt(Msg3, nb, H(j))
  → c:acceptKeyB(a,H(j),na,nb);
- Modelling state as attacker's knowledge → attacker can go back to old states of the honest participants. Why is this a sound approximation?

1. A → B: $E_B(N_A, A)$
2. B → A: $E_A(N_A, N_B)$
3. A → B: $E_B(N_B)$

$K_{ses} = h(N_A, N_B)$

9

Defining attack

- Attacker knows the session key between honest principals:
  c:acceptKeyA(H(i),H(j),na,nb) & c:na & c:nb -> c:attack;
  c:acceptKeyB(H(i),H(j),na,nb) & c:na & c:nb -> c:attack.
- If the attacker knows only one of the nonces, is that an attack?

10

Results

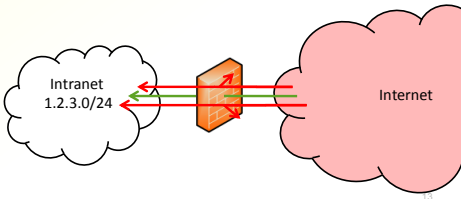
- Proverif can prove c:attack → protocol is not secure
- Manually beautified Proverif counterexample:
  encrypt(Msg3,NB,B)
  encrypt(Msg3,NB,C))
  encrypt(Msg2,(NA,NB),A)
  encrypt(Msg1,(NA,A),B)
  encrypt(Msg1,(NA,A),C)
  Selected messages only; renamed principals and nonces read from bottom up
- Fixed protocol — add B's name to Msg 2:
  2. B → A: $E_A(N_A, N_B, B)$
- Proverif output:
  RESULT goal unreachable: c:attack()

11

Stateless packet filter

Firewall

- Perimeter defence:
  - Divide the world into the safe inside (intranet) and dangerous outside (Internet)
  - Prevent anything bad from entering the inside
- Block communication that is evil, risky or just unnecessary



13

Stateless packet filter

- Allow or block IP packets based on their IP header fields and TCP/UDP port numbers
  - Fields with static locations in most IP packets: protocol (TCP/UDP/ICMP), source and destination IP address, source and destination port, TCP flags, ICMP type and code
- Packet filter is defined as a rule table
  - Linear list of rules
  - Each rule consist of conditions and an action
  - For each packet, the first matching rule is found
  - Two possible actions:
    allow (=accept, permit, bypass) or block (=drop, deny, discard), maybe also allow and log or block and log

14

Packet filter example (1)

- Example rule table: inbound email to our SMTP server 1.2.3.10

Protocol	Src IP	Src port	Dst IP	Dst port	Action	Comment
TCP	4.5.6.7	*	1.2.3.10	25	Block	Stop this spammer
TCP	*	*	1.2.3.10	25	Allow	Inbound SMTP
TCP	1.2.3.10	25	*	*	Allow	SMTP responses
*	*	*	*	*	Block	Default rule

15

Packet filter example (2)

- Allow web access from our subnet… not quite right!

Protocol	Src IP	Src port	Dst IP	Dst port	Action	Comment
TCP	1.2.3.0/24	*	*	80	Allow	Outbound HTTP requests
TCP	*	80	1.2.3.0/24	*	Allow	HTTP responses
*	*	*	*	*	Block	Default rule

- Allow only outbound connections:

Protocol	Src IP	Src port	Dst IP	Dst port	Flags	Action	Comment
TCP	1.2.3.0/24	*	*	80		Allow	Outbound HTTP requests
TCP	*	80	1.2.3.0/24	*	ACK	Allow	HTTP responses
*	*	*	*	*		Block	Default rule

(TCP packets, except the first SYN have ACK flag set)

16

Packet filter example (3)

- University lab network 1.2.3.0/24 (address 1.2.3.0, netmask 255.255.255.0)
- HTTP/Mail/DNS server 1.2.3.10

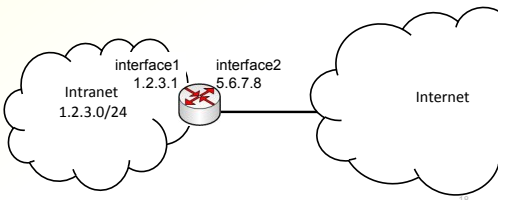
Protocol	Src IP	Src port	Dst IP	Dst port	Flags	Action	Comment
UDP	*	*	*	53		Allow	DNS queries in and out
UDP	*	53	*	*		Allow	DNS responses
TCP	5.4.3.2	*	1.2.3.10	53		Allow	DNS zone transfer
TCP	*	*	1.2.3.10	25		Allow	Inbound SMTP
TCP	*	*	1.2.3.10	80		Allow	Inbound HTTP
TCP	1.2.3.121	*	*	*		Block	Bob's test machine
TCP	*	*	1.2.3.121	*		Block	Bob's test machine
TCP	*	*	1.2.3.0/24	22		Allow	Inbound SSH
TCP	1.2.3.0/24	*	*	*		Allow	All outbound TCP
TCP	*	*	1.2.3.4/24	*	ACK	Allow	All TCP responses
*	*	*	*	*		Block	Default rule

- Is this correct? Could we limit inbound DNS queries to the server?

17

Router as packet filter

- Firewall rule table is similar to a routing table, with the option of dropping some packets
- Most routers can be used as a packet filter
  - Choice of filters may affect router throughput



18

Ingress and egress filtering

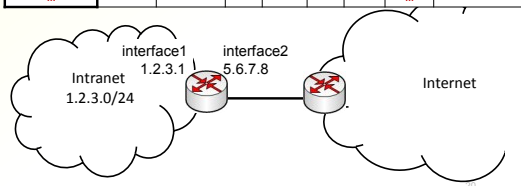
- Filter packets with topologically incorrect (probably spoofed) source IP addresses
- Ingress filtering for local network:
  - At the gateway router of a local network, drop inbound packets with source addresses that belong to the local network
- Egress filtering for local network:
  - At the gateway router of a local network, drop outbound packets with non-local source addresses
- Ingress filtering for ISP:
  - At the gateway router towards a customer, drop packets from the customer if the source address does not belong to the customer
- Egress filtering for ISP (less common):
  - At the gateway router towards a customer, drop packets to the customer if the source address belongs to the customer

19

Anti-spoofing filter example

- Filter based on input interface (partial policy only):

Input interface	Protocol	Src IP	Port	Dst IP	Port	Flags	Action	Comment
2	*	1.2.3.0/24	*	*	*		Block	Ingress filter
2	*	5.6.7.8	*	*	*		Block	Router address
1	*	1.2.3.1	*	*	*		Block	Router address
1	*	1.2.3.0/24	*	*	*		Allow	Egress filter
1	*	*	*	*	*		Block	Default rule (if1)
...							...	



20

Dynamic packet filter

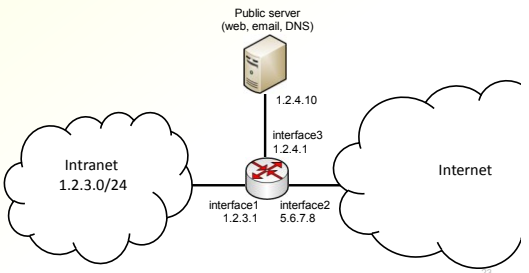
Dynamic firewall

- Stateful filter: change filtering rules based on previously seen packets
- Outbound TCP or UDP packet creates a pinhole for inbound packets of the same connection
  - Unlike stateless packet filter, can support UDP connections
- May also allow ICMP messages that match outbound traffic
- Support for special protocols:
  - FTP: firewall may sniff PORT command in FTP to open port for the inbound connections
  - X Windows

22

Typical network topology (1)

- Services accessible from the Internet are isolated to a demilitarized zone (DMZ), i.e. somewhere between the intranet and Internet



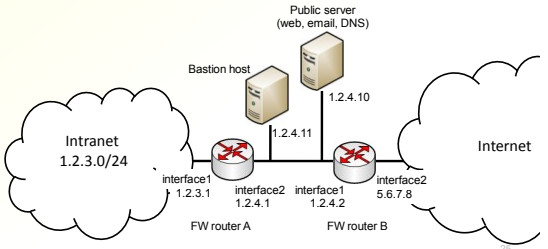
23

Input	Prot	Src IP	Port	Dst IP	Port	Other	Action	Comment
2	*	1.2.3.0/24	*	*	*		Block	Anti-spoofing
3	*	1.2.3.0/24	*	*	*		Block	Anti-spoofing
2	*	1.2.4.0/24	*	*	*		Block	Anti-spoofing
1	*	1.2.4.0/24	*	*	*		Block	Anti-spoofing
*	*	{1.2.3.1,1.2.4.1, 5.6.7.8}	*	*	*		Block	Anti-spoofing (router addr)
2	TCP	*	*	1.2.4.10	80		Allow	Access to server (HTTP)
2	TCP	*	*	1.2.4.10	443		Allow	Access to server (HTTPS)
2	TCP	*	*	1.2.4.10	25		Allow	Access to server (SMTP)
2	UDP	*	*	1.2.4.10	53		Allow	DNS query in and out
3	UDP	1.2.4.10	*	*	53		Allow	DNS query in and out
1	TCP	1.2.3.0/24	*	1.2.4.10	*		Allow, create state	Access to server from intranet
3	TCP	1.2.4.10	*	1.2.3.0/24	*	State	Allow	Responses
1	UDP	1.2.3.0/24	*	1.2.4.10	53		Allow, create state	DNS query
3	UDP	1.2.4.10	53	1.2.3.0/24	*	State	Allow	DNS response
1	*	1.2.3.0/24	*	1.2.4.0/24	*		Block	Unnecessary
3	*	1.2.4.0/24	*	1.2.3.0/24	*		Block	Unnecessary
1	*	1.2.3.0/24	*	*	*		Allow, create state	Outbound connections
2	*	*	*	*	*	State	Allow	Responses
1	TCP	1.2.3.0/24	*	{1.2.3.1,1.2.4.1, 5.6.7.8}	80		Allow, create state	Router management
-	TCP	{1.2.3.1,1.2.4.1, 5.6.7.8}	80	1.2.3.0/24	*	State	Allow	Router management
*	*	*	*	*	*		Block	Default rule

24

Typical network topology (2)

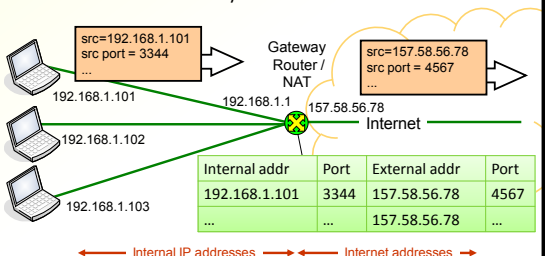
- Two-firewall configuration for isolating publicly-accessible services from the Internet
- All inbound connections use ssh and go through a hardened bastion host in the DMZ



25

NAT

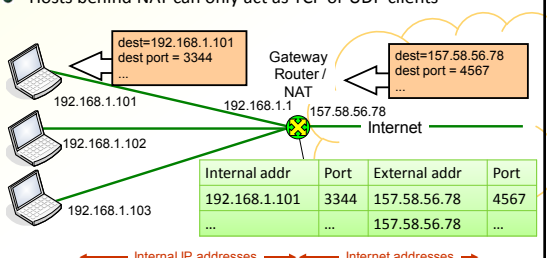
- IPv4 addresses are in short supply
- Native address translator (NAT) is a mechanisms for sharing one IPv4 address between multiple hosts
- Hosts behind NAT can only act as TCP or UDP clients



26

NAT

- IPv4 addresses are in short supply
- Native address translator (NAT) is a mechanisms for sharing one IPv4 address between multiple hosts
- Hosts behind NAT can only act as TCP or UDP clients



27

NAT as a firewall

- NAT maps internal <private IP addr, port> pairs to external <public IP addr, port> pairs and back
- NAT creates the mapping after seeing an outbound packet → a node on the intranet must initiate the connection→ NAT acts as a dynamic firewall
- NAT types:
  - Full cone NAT: NAT doesn't remember peer addresses
  - Restricted cone NAT: NAT remembers peer IP address and filters inbound packets
  - Port-restricted cone NAT: NAT remembers peer IP address and port and filters inbound packets
  - Symmetric NAT: different external port (and even address) depending the peer address and port
- Port-restricted and symmetric NATs are suitable firewalls

28

iptables

- Firewall implementation for Unix/Linux
- Complex policies can be defined as multiple chains of rules:
  - Action can be a reference to another chain
  - Provides modularity ("subroutines") for firewall policies
- Example:
  - http://www.fwbuilder.org/archives/cat_examples_of_complete_policies.html

29

Transport and application-layer firewalls

Circuit-level proxy

- Transport-layer proxy as a firewall
  - When an intranet client needs to connect to a server outside, it connects to the proxy instead
  - Proxy terminates TCP and UDP connections. Creates a second connection to the server on the Internet
  - Proxy is simpler than a host, hardened against attacks, and filters and normalizes connections
- SOCKS management protocol between client and firewall
  - Client requests new connections
  - Authentication and authorization of client requests, e.g. GSSAPI
  - Error messages to client
  - Supported by most web browsers
  - Implemented in Microsoft Firewall Client and ISA Server
- Firewall router can be set up to forward only some connections to the proxy for closer inspection

31

Application-level firewall

- Application-level firewall filters application data
  - E.g. email gateway, intercepting web proxy
  - Need to implement the entire application protocol
- Telephone call blocking and barring vs. wiretapping
- Encrypted data cannot be filtered → what do you do?
- Are latest applications and features supported?

32

Firewall issues

33

Why filter outbound connections

- Security:
  - Prevent people from accessing untrusted services or dangerous content
  - Prevent compromised machines from spreading viruses to the Internet, phishing etc.
- Cost:
  - Businesses and other organizations are charged by megabyte → block access to P2P, VoIP
- Productivity:
  - How do employees spend their time?
- Liability:
  - Does free Internet access by employees or visitors expose the company to legal risks?

34

Firewall traversal

- Network admins prefer to block traffic by default → New applications and protocols will not work
- New applications will not gain popularity if an administrative decision is needed at each site → application developers (and users) do their best to circumvent firewalls
  - Web services over port 80, everything over port 443
  - Skype, P2P protocols
- Discussion: Should all new network applications be standardized and get a port number from IANA, so that they can be filtered by the firewall?

35

Debugging firewall rules

- Firewall rules are difficult to configure
  - Order of rules matters → fragile configurations
  - Configuration language, its exact semantics and expressiveness varies between implementations
  - Stateless packet filters have limited expressive power
- Performance depends on router hardware
  - Routing may become slower when filtering is enabled, or when specific filters are deployed. What is processed in hardware?
- Redundancy may be a clue to errors, but not always:
  - Rule is shadowed if another rule above it prevents it from ever being triggered. Is this intentional?
  - Overlapping rules match some of the same packets. Do they specify the same action or different ones?
  - In a network with multiple firewalls, do you want to block packets that are already blocked by another firewall?

36

Firewall limitations

- May prevent people from doing their work
  - Try to convince a network admin to open a pinhole for your server
- Network admins are often reluctant to change firewall policies in case something breaks
- Makes network diagnostics harder
- Firewall configuration errors are common
- Coarse-grained filtering for efficient routing and administration
- Perimeter defence is ineffective in large networks
  - There are always some compromised nodes inside
- Unfiltered ingress routes:
  - Dial-up modem connections in and out
  - Unauthorized access points
- Laptops move in and out of the intranet
- Security of home gateways and other network devices is questionable
- Most applications now use TCP port 80 or 443, or use other clever tricks to traverse firewalls

37

Exercises

- Why cannot ingress filtering ever stop all IP spoofing attacks?
- Do you find any mistakes or shortcomings in the firewall policy examples of this lecture?
- Find out what kind of firewall capabilities your home gateway router/NAT has.
- Find the firewall configuration of a small network. Try to understand each line of the policy. Have compromises on security been made to achieve better performance, to make management easier, or because of limitations in the router?
- Write firewall policies for the Network topology example (2) in an earlier slide. What compromises will you have to make if the firewalls are stateless packet filters and do not support filtering based on the input interface.
- Stateless firewall typically allows all inbound TCP packets with the ACK flag set. On a 1 GB/s network, how difficult is it for external attackers to spoof some TCP packets that match the sequence numbers of an intranet TCP connection?

38

Exercises

- How to attack the Needham-Schroeder secret-key protocol if the encryption is no integrity-protecting, e.g. $E_K(M)$ = AES-ECB$_K$(M) ?
- Read about the Yahalom and Otway-Rees protocols. Can you find any flaws by yourself?
- Model the Needham-Schroeder shared-key protocol in Proverif
- How would you model identity or DoS protections with a tool like Proverif? (This is a difficult question.)

39