Firewalls

Outline

- What are firewalls?
- Types of Firewalls
- Building a simple firewall using Netfilter
- Iptables firewall in Linux
- Stateful Firewall
- Application Firewall
- Evading Firewalls

Firewalls

- A part of computer system or network designed to stop unauthorized traffic flowing from one network to another.
- Separate trusted and untrusted components of a network.
- Differentiate networks within a trusted network.
- Main functionalities are filtering data, redirecting traffic and protecting against network attacks.

Requirements of a firewall

- All the traffic between trust zones should pass through firewall.
- Only authorized traffic, as defined by the security policy, should be allowed to pass through.
- The firewall itself must be immune to penetration, which implies using a hardened system with secured Operating Systems.

Firewall Policy

- User control: Controls access to the data based on the role of the user who is attempting to access it. Applied to users inside the firewall perimeter.
- Service control: Controls access by the type of service offered by the host. Applied on the basis of network address, protocol of connection and port numbers.
- Direction control: Determines the direction in which requests may be initiated and are allowed to flow through the firewall. It tells whether the traffic is “inbound” (From the network to firewall) or vice-versa “outbound”

Firewall actions

Accepted: Allowed to enter the connected network/host through the firewall.

Denied: Not permitted to enter the other side of firewall.

Rejected: Similar to “Denied”, but tells the source about this decision through ICMP packet.

Ingress filtering: Inspects the incoming traffic to safeguard an internal network and prevent attacks from outside.

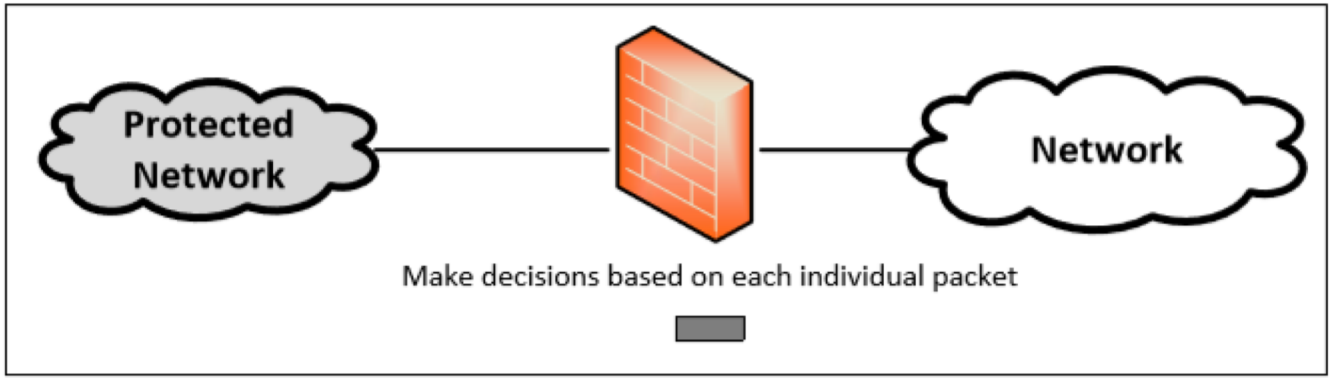
Egress filtering: Inspects the outgoing network traffic and prevent the users in the internal network to reach out to the outside network. For example like blocking social networking sites in school

Types of filters

Depending on the mode of operation, there are three types of firewalls :

- Packet Filter Firewall
- Stateful Firewall
- Application/Proxy Firewall

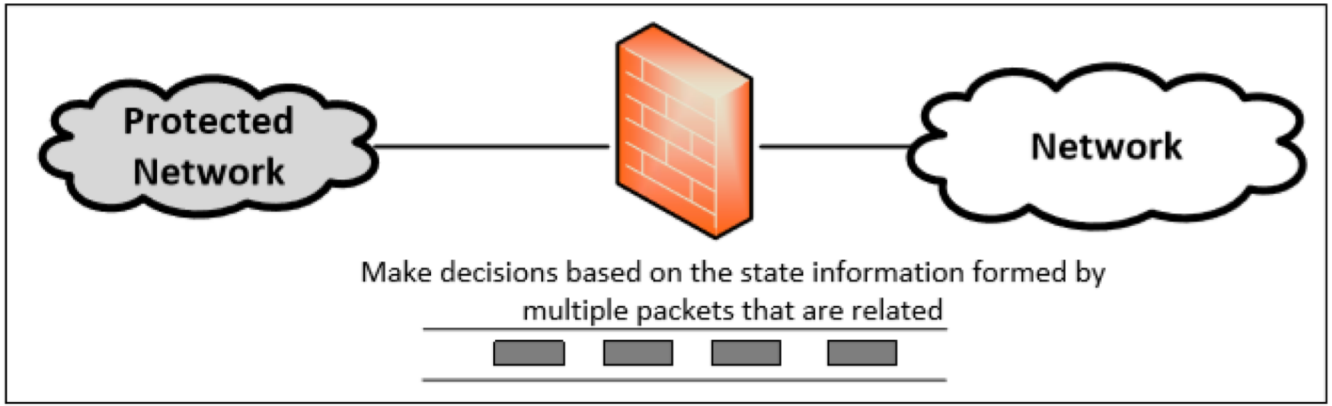
Packet Filter Firewall



- Doesn’t pay attention to if the packet is a part of existing stream or traffic.
- Doesn’t maintain the states about packets. Also called Stateless Firewall.

- Controls traffic based on the information in packet headers, without looking into the payload that contains application data.

Stateful Firewall

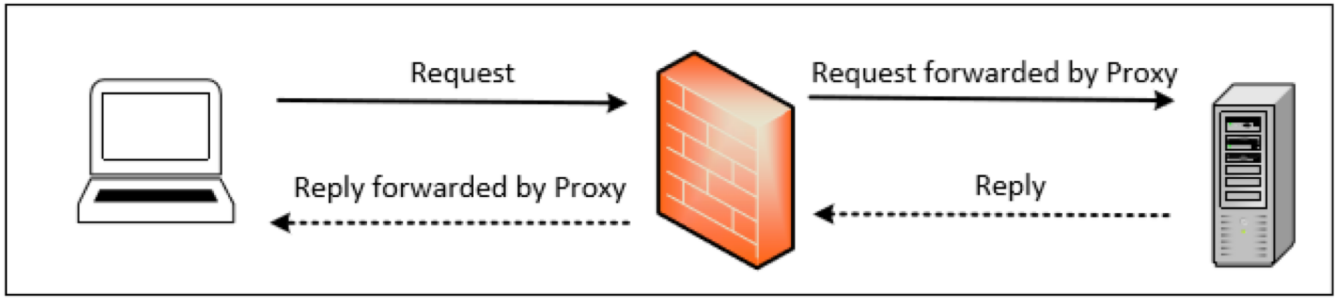


- Example : Connections are only allowed through the ports that hold open connections.

- Tracks the state of traffic by monitoring all the connection interactions until is closed.

- Connection state table is maintained to understand the context of packets.

Application/Proxy Firewall



- The client's connection terminates at the proxy and a separate connection is initiated from the proxy to the destination host.
- Data on the connection is analyzed up to the application layer to determine if the packet should be allowed or rejected.

- Controls input, output and access from/to an application or service.
- Acts an intermediary by impersonating the intended recipient.

Building a Firewall using Netfilter

Packet filter firewall implementation in Linux

- Packet filtering can be done inside the kernel.
- Need changes in the kernel
- Linux provides two mechanisms to achieve this :

Netfilter: Provides hooks at critical points on the packet traversal path inside Linux Kernel.

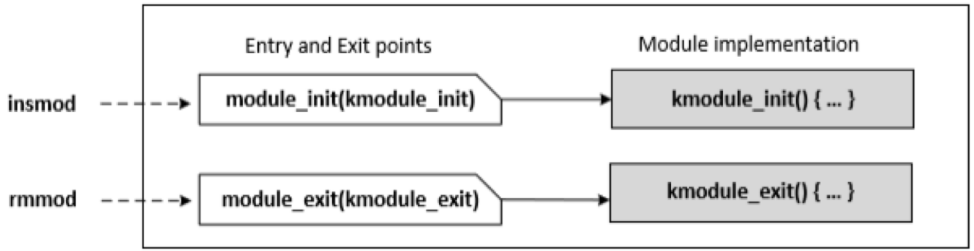
Loadable Kernel Modules: Allow privileged users to dynamically add/remove modules to the kernel, so there is no need to recompile the entire kernel.

Loadable Kernel Modules

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>

static int kmodule_init(void) {
   printk(KERN_INFO "Initializing this module\n");
   return 0;
}

static void kmodule_exit(void) {
   printk(KERN_INFO "Module cleanup\n");
}

module_init(kmodule_init);        ①
module_exit(kmodule_exit);        ②

MODULE_LICENSE("GPL");
```

Specify an initialization function that will be invoked when the kernel module is inserted.

Specify a cleanup function that will be invoked when the kernel module is removed.



Compiling Kernel Modules

```
obj-m += kMod.o
all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

```
$ make
make -C /lib/modules/3.5.0-37-generic/build
    M=/home/seed/labs/firewall/lkm modules
make[1]: Entering directory '/usr/src/linux-headers-3.5.0-37-generic'
  CC [M]  /home/seed/labs/firewall/lkm/kMod.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/seed/labs/firewall/lkm/kMod.mod.o
  LD [M]  /home/seed/labs/firewall/lkm/kMod.ko
make[1]: Leaving directory '/usr/src/linux-headers-3.5.0-37-generic'
```

Makefile

M: Signifies that an external module is being built and tells the build environment where to place the built module file.

C: Specify the directory of the library files for the kernel source.

Installing Kernel Modules

```
// Insert the kernel module into the running kernel.
$ sudo insmod kMod.ko

// List kernel modules
$ lsmod | grep kMod
kMod                   12453  0

// Remove the specified module from the kernel.
$ sudo rmmod kMod
```

```
$ dmesg
......
[65368.235725] Initializing this module
[65499.594389] Module cleanup
```

In the sample code, we use `printk()` to print out messages to the kernel buffer. We can view the buffer using `dmesg`.

Netfilter

- Netfilter hooks are rich packet processing and filtering framework.
- Each protocol stack defines a series of hooks along the packet’s traversal path in the stack.
- Developers can use LKMs to register callback functions to these hooks.
- When a packet arrives at each of these hooks, the protocol stack calls the netfilter framework with the packet and hook number.
- Netfilter checks if any kernel module has registered a callback function at this hook.
- Each registered module will be called, and they are free to analyze or manipulate the packet and return the verdict on the packet.

Netfilter: Verdict on Packets (Return Values)

`NF_ACCEPT`: Let the packet flow through the stack.

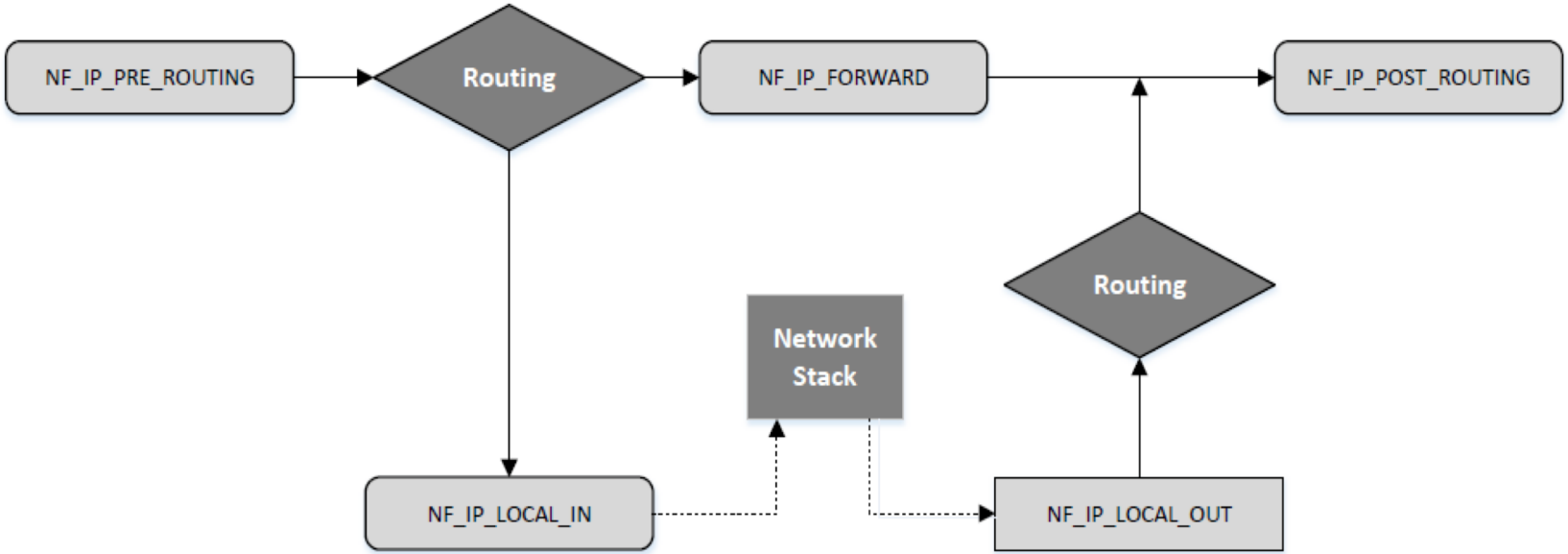
`NF_DROP`: Discard the packet.

`NF_QUEUE`: Pass the packet to the user space via nf_queue facility.

`NF_STOLEN`: Inform the netfilter to forget about this packet, The packet is further processed by the module.

`NF_REPEAT`: Request the netfilter to call this module again.

Netfiler Hooks for IPv4



Implementing a Simple Packet Filter Firewall

```
unsigned int telnetFilter(unsigned int hooknum, struct sk_buff *skb,
      const struct net_device *in, const struct net_device *out,
      int (*okfn)(struct sk_buff *)) {
  struct iphdr *iph;
  struct tcphdr *tcph;

  iph = ip_hdr(skb);
  tcph = (void *)iph+iph->ihl*4;

  if (iph->protocol == IPPROTO_TCP && tcph->dest == htons(23)) {
    printk(KERN_INFO "Dropping telnet packet to %d.%d.%d.%d\n",
    ((unsigned char *)&iph->daddr)[0],
    ((unsigned char *)&iph->daddr)[1],
    ((unsigned char *)&iph->daddr)[2],
    ((unsigned char *)&iph->daddr)[3]);
    return NF_DROP;
  } else {
    return NF_ACCEPT;
  }
}
```

The entire packet is provided here.

The filtering logic is hardcoded here. Drop the packet if the destination TCP port is 23 (telnet)

Decisions

Implementing a Simple Packet Filter Firewall

```
int setUpFilter(void) {
   printk(KERN_INFO "Registering a Telnet filter.\n");
   telnetFilterHook.hook = telnetFilter;
   telnetFilterHook.hooknum = NF_INET_POST_ROUTING;
   telnetFilterHook.pf = PF_INET;
   telnetFilterHook.priority = NF_IP_PRI_FIRST;

   // Register the hook.
   nf_register_hook(&telnetFilterHook);
   return 0;
}

void removeFilter(void) {
   printk(KERN_INFO "Telnet filter is being removed.\n");
   nf_unregister_hook(&telnetFilterHook);
}

module_init(setUpFilter);
module_exit(removeFilter);
```

Hook this callback function

Use this Netfilter hook

Register the hook

Testing Our Firewall

```
$ sudo insmod telnetFilter.ko
$ telnet 10.0.2.5
Trying 10.0.2.5...
telnet: Unable to connect to remote host: ...    ← Blocked!
$ dmesg
......
[1166456.149046] Registering a Telnet filter.
[1166535.962316] Dropping telnet packet to 10.0.2.5
[1166536.958065] Dropping telnet packet to 10.0.2.5

// Now, let's remove the kernel module

$ sudo rmmod telnetFilter
$ telnet 10.0.2.5
telnet 10.0.2.5
Trying 10.0.2.5...
Connected to 10.0.2.5.
Escape character is '^]'.
Ubuntu 12.04.2 LTS
ubuntu login:              ← Succeeded!
```

Iptables Firewall in Linux

- Iptables is a built-in firewall based on netfilter.
- Kernel part: Xtables
- User-space program: iptables
- Usually, iptables refer to both kernel and user space programs.
- Rules are arranged in hierarchical structure as shown in the table.

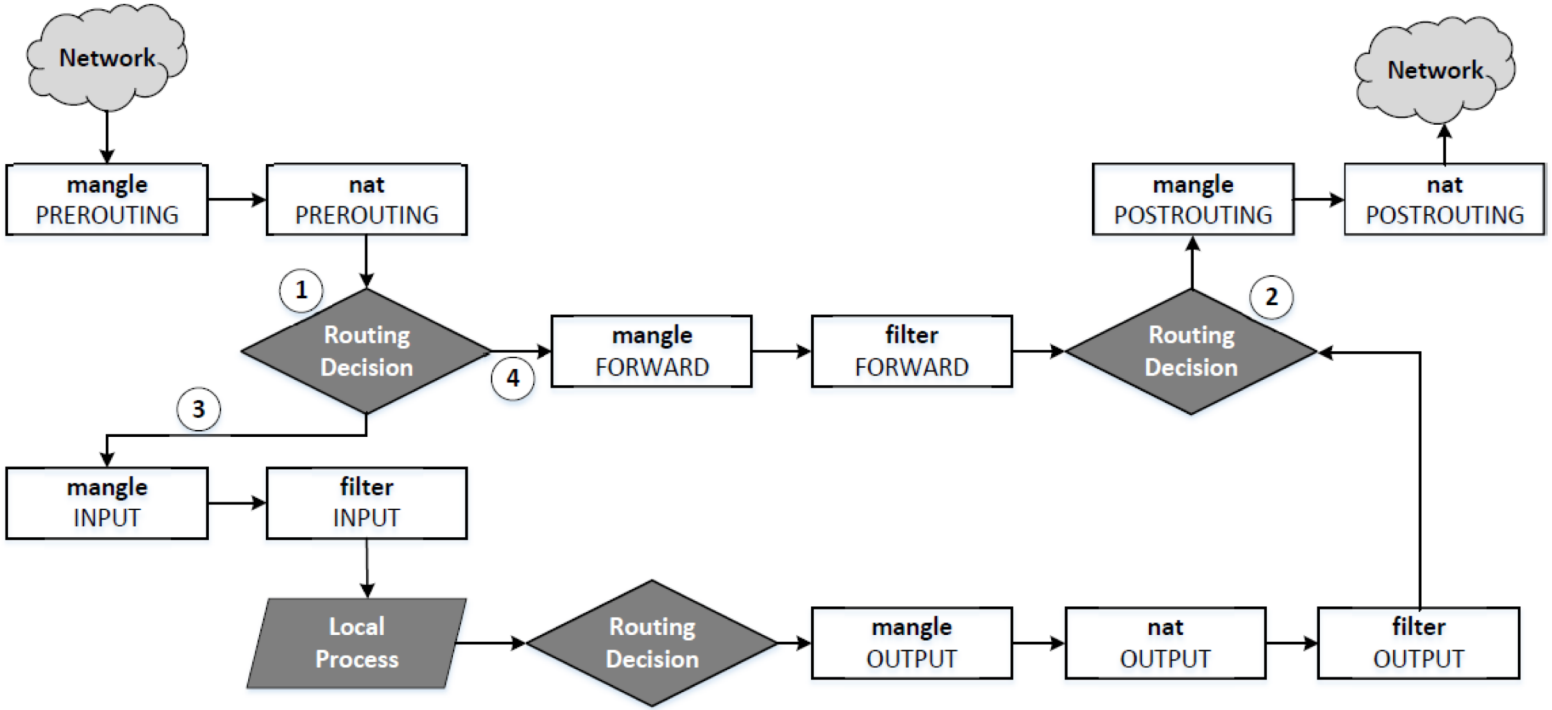
Table	Chain	Functionality
filter	INPUT FORWARD OUTPUT	Packet filtering
nat	PREROUTING INPUT OUTPUT POSTROUTING	Modifying source or destination network addresses
mangle	PREROUTING INPUT FORWARD OUTPUT POSTROUTING	Packet content modification

Iptables Firewall - Structure

- Each table contains several chains, each of which corresponds to a netfilter hook.
- Each chain indicates where its rules are enforced.
  - Example : Rules on FORWARD chain are enforced at NF_IP_FORWARD hook and rules on INPUT chain are enforced at NF_IP_LOCAL_IN hook.

- Each chain contains a set of firewall rules that will be enforced.
- User can add rules to the chains.
  - Example : To block all incoming telnet traffic, add a rule to the INPUT chain of the filter table

Traversing Chains and Rule Matching



Traversing Chains and Rule Matching

1 - Decides if the final destination of the packet is the local machine
3 - Packet traverses through INPUT chains
4 - Packet traverses through FORWARD chains
2 - Decides from which of the network interface to send out outgoing packets

As a packet traverses through each chain, rules on the chain are examined to see whether there is a match or not. If there is a match, the corresponding target action is executed: ACCEPT, DROP or jumping to user-defined chain.

Traversing Chains and Rule Matching

Example: Increase the TTL field of all packets by 5.

Solution: Add a rule to the mangle table and choose a chain provided by netfilter hooks. We choose PREROUTING chain so the changes can be applied to all packets, regardless they are for the current host or for others.

```
// -t mangle = Add this to 'mangle' table
// -A PREROUTING = Append this rule to PREROUTING chain

iptables -t mangle -A PREROUTING -j TTL --ttl-inc 5
```

Iptables Extension

Iptables functions can be extended using modules also called as extensions.

Two Examples:

Conntrack: To specify rules based on connections to build stateful firewalls.

Owner: To specify rules based on user ids. Ex: To prevent user Alice from sending out telnet packets. Owner module can match packets based on the user/group id of the process that created them. This works only for OUTPUT chain (outgoing packets) as it is impossible to find the user ids for INPUT chain(incoming packets).

Iptables Extension: Block a Specific User

```
seed$ sudo iptables -A OUTPUT -m owner --uid-owner seed -j DROP
seed$ telnet 10.0.2.5
Trying 10.0.2.5...
telnet: Unable to connect to remote host: ...   ← telnet is blocked!

seed$ su bob
Password:
bob$ telnet 10.0.2.5
Trying 10.0.2.5...
Connected to 10.0.2.5.
Escape character is '^]'.
Ubuntu 12.04.2 LTS
ubuntu login:              ← telnet works!
```

This rule drops the packets generated by any program owned by user seed. Other users are not affected.

Building a Simple Firewall

- Flush all existing firewall configurations
- Default policy is set to ACCEPT before all the rules.

```
// Set up all the default policies to ACCEPT packets.
$ sudo iptables -P INPUT ACCEPT
$ sudo iptables -P OUTPUT ACCEPT
$ sudo iptables -P FORWARD ACCEPT

// Flush all existing configurations.
$ sudo iptables -F
```

Building a Simple Firewall

- Rule on INPUT chain to allow TCP traffic to ports 22 and 80

```
// Allow all incoming TCP packets bound to destination port 22.
// -A INPUT: Append to existing INPUT chain rules.
// -p tcp: Select TCP packets
// -dport 22: Select packets with destination port 22.
// -j ACCEPT: Accept all the packets that are selected.
$ sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT

// Similarly, accept all packets bound to destination port 80.
$ sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT
```

- Rule on OUTPUT chain to allow all outgoing TCP traffic

```
// Allow all outgoing TCP traffic.
// -A OUTPUT: Append to existing OUTPUT chain rules.
// -p tcp: Apply on TCP protocol packets
// -m tcp: Further apply matching rules defined in 'tcp' module.
// -j ACCEPT: Let the selected packets through.

$ sudo iptables -A OUTPUT -p tcp -m tcp -j ACCEPT
```

Building a Simple Firewall

- Allow the use of the loopback interface.

```
// -I INPUT 1 : Insert a rule in the 1st position of the INPUT chain.
// -i lo : Select packets bound for the loopback (lo) interface.
// -j ACCEPT: Accept all the packets that are selected.

$ sudo iptables -I INPUT 1 -i lo -j ACCEPT
```

- Allow DNS queries and replies to pass through.

```
// Allow DNS queries and replies to pass through.

$ sudo iptables -A OUTPUT -p udp --dport 53 -j ACCEPT
$ sudo iptables -A INPUT -p udp --sport 53 -j ACCEPT
```

Building a Simple Firewall

```
seed@ubuntu:$ sudo iptables -L
Chain INPUT (policy DROP)
target     prot opt source      destination
ACCEPT     tcp  --  anywhere    anywhere      tcp dpt:ssh
ACCEPT     tcp  --  anywhere    anywhere      tcp dpt:http
ACCEPT     udp  --  anywhere    anywhere      udp spt:domain

Chain FORWARD (policy DROP)
target     prot opt source      destination

Chain OUTPUT (policy DROP)
target     prot opt source      destination
ACCEPT     tcp  --  anywhere    anywhere     tcp
ACCEPT     udp  --  anywhere    anywhere     udp dpt:domain
```

These are all the rules we have added

```
// Setting default filter policy to DROP.
$ sudo iptables -P INPUT DROP
$ sudo iptables -P OUTPUT DROP
$ sudo iptables -P FORWARD DROP
```

Change the default policy to DROP so that only our configurations on firewall work.

Building a Simple Firewall: Testing

```
$ telnet 10.0.2.6        ← Our firewall is running on 10.0.2.6.
Trying 10.0.2.6...
telnet: Unable to connect to remote host: ...        ← Blocked!
$ wget 10.0.2.6
--2017-01-25 18:31:41--  http://10.0.2.6/
Connecting to 10.0.2.6:80... connected.
HTTP request sent, awaiting response... 200 OK        ← Succeeded!
```

- To test our firewall, make connection attempts from a different machine.
- Firewall drops all packets except the ones on ports 80(http) and 22(ssh).
- Telnet connection made on port 23 failed to connect, but wget connection on port 80 succeeded.

Stateful Firewall using Connection Tracking

- A stateful firewall monitors incoming and outgoing packets over a period of time.
- Records attributes like IP address, port numbers, sequence numbers. Collectively known as connection states.
- A connection state, in context of a firewall signifies whether a given packet is a part of an existing flow or not.
- Hence, it is applied to both connection-oriented (TCP) and connectionless protocols (UDP and ICMP).

Connection Tracking Framework in Linux

- nf_conntrack is a connection tracking framework in Linux kernel built on the top of netfilter.
- Each incoming packet is marked with a connection state as described:
  - NEW: The connection is starting and packet is a part of a valid sequence. It only exists for a connection if the firewall has only seen traffic in one direction.
  - ESTABLISHED: The connection has been established and is a two-way communication.
  - RELATED: Special state that helps to establish relationships among different connections. E.g., FTP Control traffic and FTP Data traffic are related.
  - INVALID : This state is used for packets that do not follow the expected behavior of a connection.
- iptables can use nf_conntrack to build stateful firewall rules.

Example: Set up a Stateful Firewall

```
// -A OUTPUT: Append to existing OUTPUT chain rules.
// -p tcp: Apply on TCP protocol packets.
// -m conntrack: Apply the rules from conntrack module.
// --ctsate ESTABLISHED,RELATED: Look for traffic in ESTABLISHED or
    RELATED states.
// -j ACCEPT: Let the selected packets through.

$ sudo iptables -A OUTPUT -p tcp -m conntrack --ctstate
    ESTABLISHED,RELATED -j ACCEPT
```

- To set up a firewall rule to only allow outgoing TCP packets if they belong to an established TCP connection.
- We only allow ssh and http connection and block all the outgoing TCP traffic if they are not part of an ongoing ssh or http connection.
- We will replace the earlier rule with this one based on the connection state.

Application/Proxy Firewall and Web Proxy

- Inspects network traffic up to the application layer.
- Typical implementation of an application firewall is a proxy (application proxy)
- Web proxy: To control what browsers can access.
- To set up a web proxy in a network, we need to ensure that all the web traffic goes through the proxy server by:
    - Configuring each host computer to redirect all the web traffic to the proxy. (Browser's network settings or using iptables)
    - Place web proxies on a network bridge that connects internal and external networks.

Application/Proxy Firewall and Web Proxy

- Proxy can also be used to evade egress filtering.
  - If a firewall conducts packet filtering based on destination address, we can evade this firewall by browsing the Internet using web proxy.
  - The destination address will be modified to the proxy server which defeats the packet filtering rules of the firewall.

- Anonymizing Proxy: One can also use proxies to hide the origin of a network request from servers. As the servers can only see the traffic after it passes through proxies, source IP will be the proxy's and actual origin is hidden.

Evading Firewalls

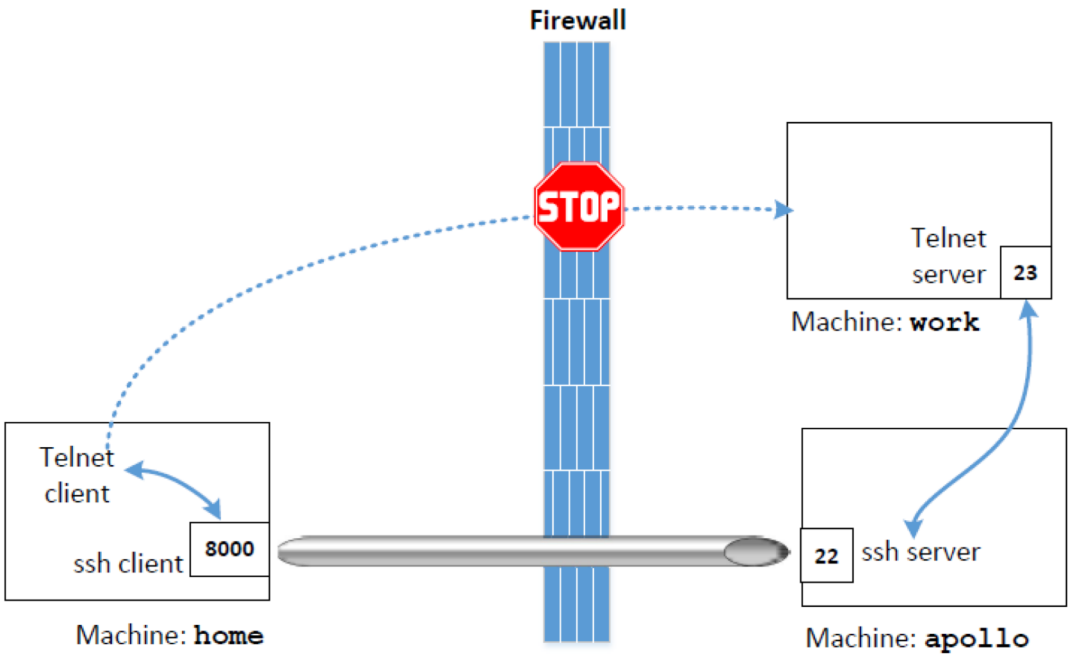
- SSH Tunneling
- Dynamic Port Forwarding
- Virtual Private Network

SSH Tunneling to Evade Firewalls

Scenario :

We are working in a company and need to telnet to a machine called “work”. Sometimes as we work from home, we need to telnet from machine “home” to “work”. However, the company’s firewall blocks all incoming traffic which makes telnet from “home” impossible.The company’s firewall does allow ssh traffic to reach its internal machine “apollo”, where we have an account. How can we use this machine to evade the firewall?

SSH Tunneling to Evade Firewalls



- Establish a ssh tunnel between “home” and “apollo”.
- On the “home” end, the tunnel receives TCP packets from the telnet client.
- It forwards the TCP data to “apollo” end, from where the data is out in another TCP packet which is sent to machine “work”.
- The firewall can only see the traffic between “home” and “apollo” and not from “apollo” to “work”. Also ssh traffic is encrypted.

SSH Tunneling to Evade Firewalls

```
// Establish the tunnel from Machine home to Machine apollo
$ ssh -L 8000:work:23  apollo

// Telnet to Machine work from Machine home
$ telnet localhost 8000
```

- Establish an ssh tunnel from “home” to “apollo”. This tunnel will forward TCP data received on 8000 on “home” to port 23 on work.
- After establishing the tunnel, telnet to the 8000, and the telnet traffic will be forwarded host work via the ssh tunnel.

SSH Tunneling to Evade Firewalls

Scenario : We are working in a company and working on a machine called “work”. We would like to visit Facebook, but the company has blocked it to prevent employees from getting distracted. We use an outside machine “home” to bypass such a firewall. How can we bypass it?

```
$ ssh -L 8000:www.facebook.com:80  home
```

- We establish an ssh tunnel from “work” to “home”.
- After establishing the tunnel, we can type “localhost:8000” in our browser.
- The tunnel will forward our HTTP requests to Facebook via home.
- The firewall can only see the ssh traffic between “work” and “home” and not the actual web traffic between “work” and “Facebook”.

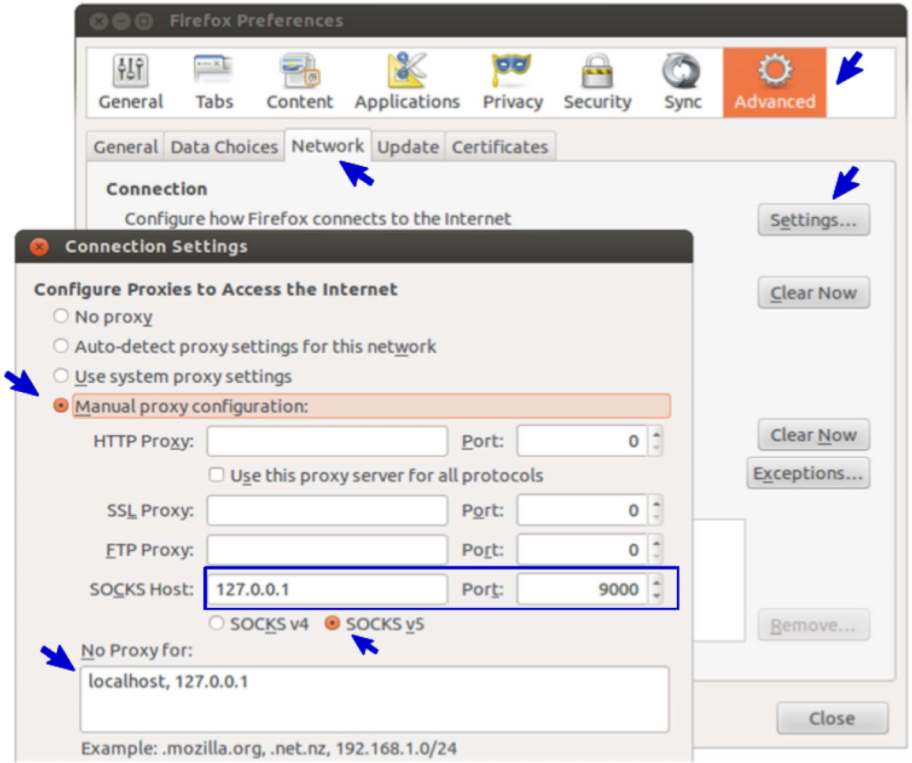
Dynamic Port Forwarding

```
$ ssh -D 9000 -C home
```

- This command establishes an ssh tunnel between localhost (port 9000) and the machine “home”. Here we do not specify the destination for the port forwarding.
- So, we configure the browser in such a way that all the requests should go through localhost:9000, treating it as a proxy.
- Dynamic port forwarding that we set up using ssh is a ***SOCKS proxy***.
- Once the browser is configured, we can type URL of any blocked site which will connect to ssh proxy at port 9000 on the localhost.
- ssh will send the TCP data over the tunnel to the machine “home” which will communicate with the blocked site.

Dynamic Port Forwarding

The client software must have a native SOCKS support to use SOCKS proxies.



Using VPN to Evade Firewall

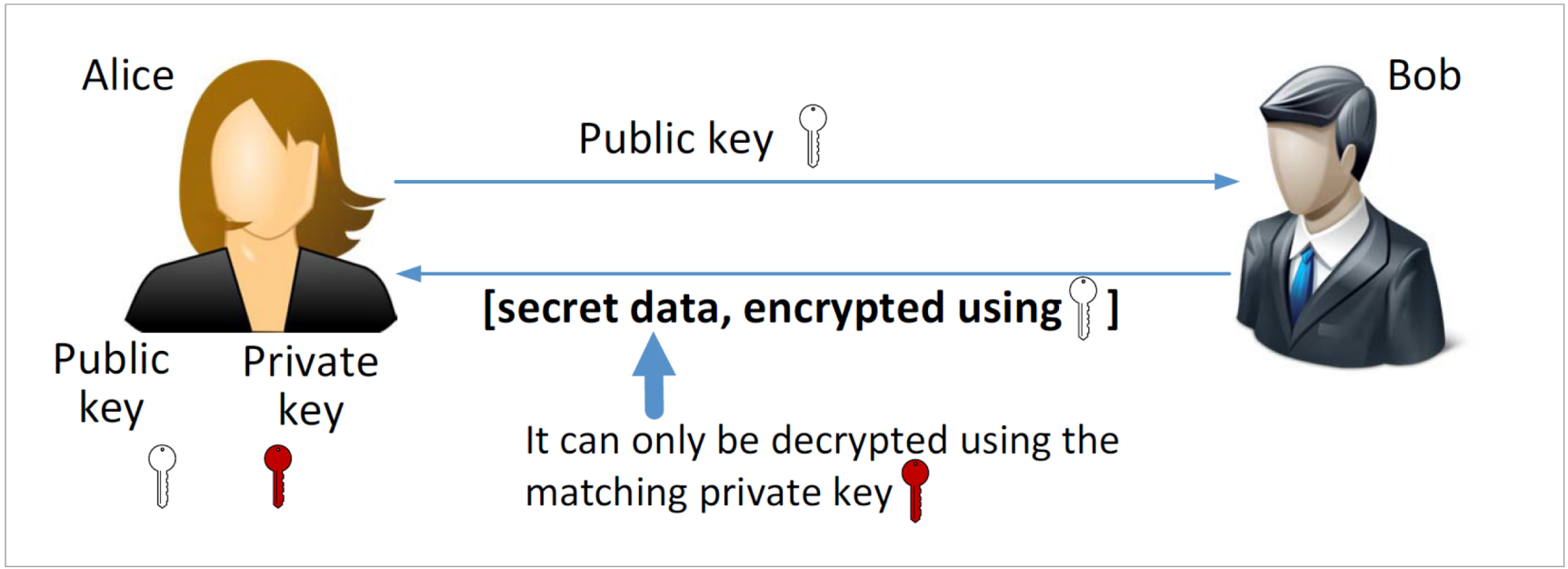
Using VPN, one can create a tunnel between a computer inside the network and another one outside. IP packets can be sent using this tunnel. Since the tunnel traffic is encrypted, firewalls are not able to see what is inside this tunnel and cannot conduct filtering. This topic is covered in detail late in VPN topic.

Summary

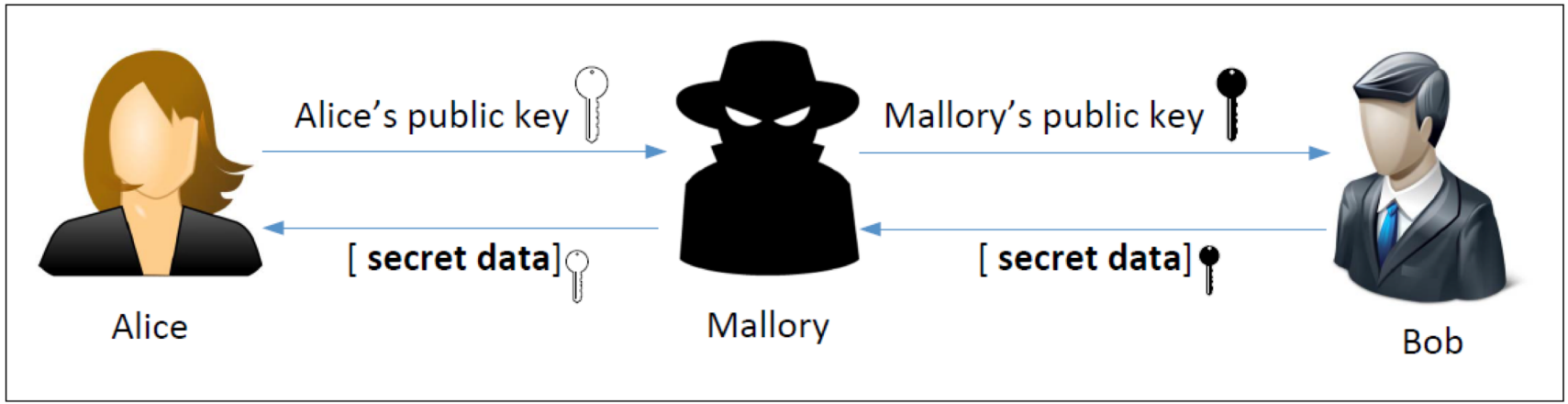
- The concept of firewall
- Implement a simple firewall using netfilter
- Using iptables to configure a firewall
- Stateful firewalls and web proxy
- Bypassing firewalls

Public Key Infrastructure

Public Key Cryptography



Man-in-the-Middle (MITM) Attack



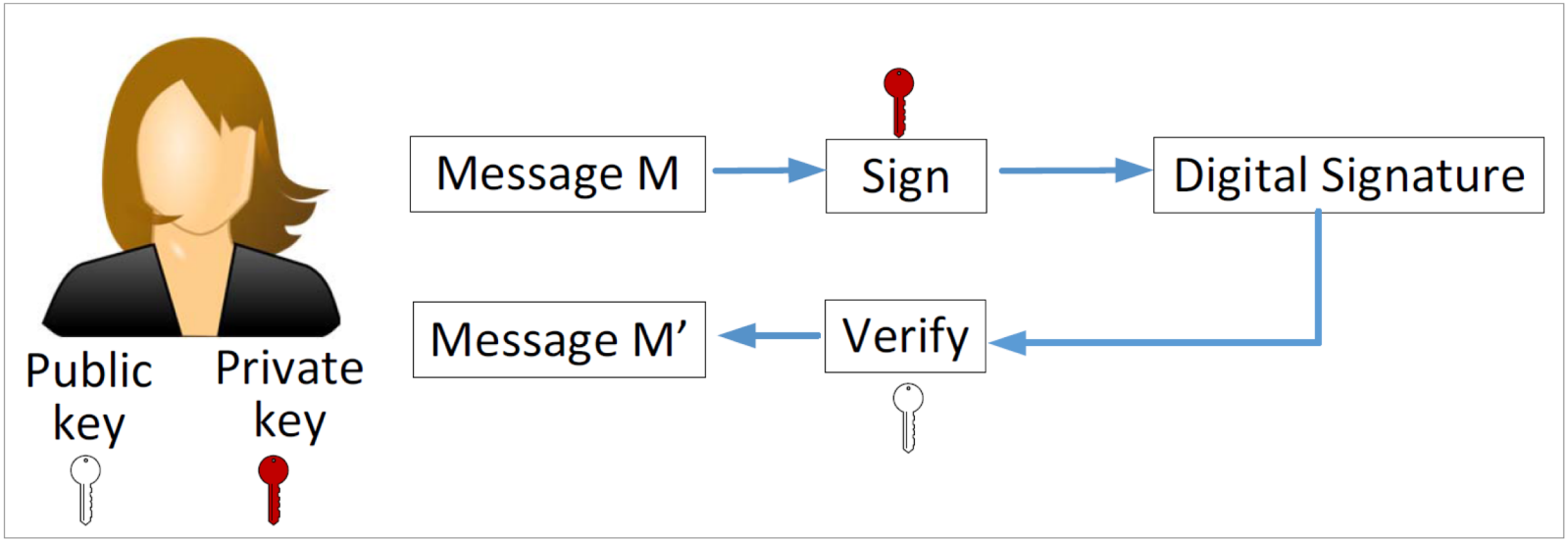
What Is the Fundamental Problem?

Fundamental Problem: Bob has no way to tell whether the public key he has received belongs to Alice or not.

Solution:

- Find a trusted party to verify the identity
- Bind an identity to a public key in a certificate
- The certificate cannot be forged or tampered with (using digital signature)

Digital Signature



- If the signature is not tampered with, M' will be the same as M
- Only Alice can sign (she has the private key)
- Everybody can verify (public key is known publically)

Defeating MITM Attacks using Digital Signature

- Alice needs to go to a trusted party to get a certificate.
- After verifying Alice's identity, the trusted party issues a certificate with Alice's name and her public key.
- Alice sends the entire certificate to Bob.
- Bob verifies the certificate using the trusted party's public key.
- Bob now knows the true owner of a public key.

Public Key Infrastructure

- **Certificate Authority (CA):** a trusted party, responsible for verifying the identity of users, and then bind the verified identity to a public keys.
- **Digital Certificates:** A document certifying that the public key included inside does belong to the identity described in the document.
  - X.509 standard

Digital Certificate

- Let's get paypal's certificates

```
$ openssl s_client -showcerts -connect www.paypal.com:443 </dev/null
```

```
-----BEGIN CERTIFICATE-----
MIIHWTCCBkGgAwIBAgIQLNGVEFQ30N5KOSAFavbCfzANBgkqhkiG9w0BAQsFADB3
MQswCQYDVQQGEwJVUzEdMBsGA1UEChMUU3ltYW50ZWMgQ29ycG9yYXRpb24xHzAd
... (omitted) ...
GN/QMQ3a55rjwNQnA3s2WWuHGPaE/jMG17iiL2O/hUdIvLE9+wA+fWrey5//74xl
NeQitYiySDIepHGnng==
-----END CERTIFICATE-----
```

- Save the above data to paypal.pem, and use the following command decode it (see next slide)

```
$ openssl x509 -in paypal.pem -text -noout
```

Example of X.509 Certificate (1st Part)

The CA's identity (Symantec)

The owner of the certificate (paypal)

```
Certificate:
Data:
  Serial Number:
          2c:d1:95:10:54:37:d0:de:4a:39:20:05:6a:f6:c2:7f
  Signature Algorithm: sha256WithRSAEncryption
  Issuer: C=US, O=Symantec Corporation, OU=Symantec Trust Network,
        CN=Symantec Class 3 EV SSL CA - G3
  Validity
     Not Before: Feb  2 00:00:00 2016 GMT
     Not After : Oct 30 23:59:59 2017 GMT
  Subject: 1.3.6.1.4.1.311.60.2.1.3=US/
          1.3.6.1.4.1.311.60.2.1.2=Delaware/
          businessCategory=Private Organization/
          serialNumber=3014267, C=US/
          postalCode=95131-2021, ST=California,
          L=San Jose/street=2211 N 1st St,
          O=PayPal, Inc., OU=CDN Support, CN=www.paypal.com
```

Example of X.509 Certificate (2nd Part)

Public key

CA's signature

```
  Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
         Public-Key: (2048 bit)
         Modulus:
           00:da:43:c8:b3:a6:33:5d:83:c0:63:14:47:fd:6b:22:bd:
           bf:4e:a7:43:11:55:eb:20:8b:e4:61:13:ee:de:fe:c6:e2:
           ... (omitted) ...
           7a:15:00:c5:01:69:b5:10:16:a5:85:f8:fd:07:84:9a:c9:
         Exponent: 65537 (0x10001)
Signature Algorithm: sha256WithRSAEncryption
4b:a9:64:20:cc:77:0b:30:ab:69:50:d3:7f:de:dc:7c:e2:fb:93:84:fd:
78:a7:06:e8:14:03:99:c0:e4:4a:ef:c3:5d:15:2a:81:a1:b9:ff:dc:3a:
... (omitted) ...
fb:00:3e:7d:6a:de:cb:9f:ff:ef:8c:65:35:e4:22:b5:88:b2:48:32:1e:
```

The Core Functionalities of CA

- **Verify the subject**
  - Ensure that the person applying for the certificate either owns or represents the identity in the subject field.

- **Signing digital certificates**
  - CA generates a digital signature for the certificate using its private key.
  - Once the signature is applied, the certificate cannot be modified.
  - Signatures can be verified by anyone with the CA’s public key.

Being a Certificate Authority

- Let’s go through the process
  - How a CA issues certificates
  - How to get a certificate from a CA
  - How to set up a web server using a certificate

CA Setup

- Our CA will be called ModelCA
  - We need to set up the following for ModelCA:
    - Generate public/private key pair
    - Create a X.509 certificate (who is going to sign it?)
    - We assume ModelCA is a root CA, so it is going to sign the certificate itself, i.e. self-signed.
  -

```
$ openssl req -x509 -newkey rsa:4096 -sha256 -days 3650
              -keyout modelCA_key.pem -out modelCA_cert.pem
```

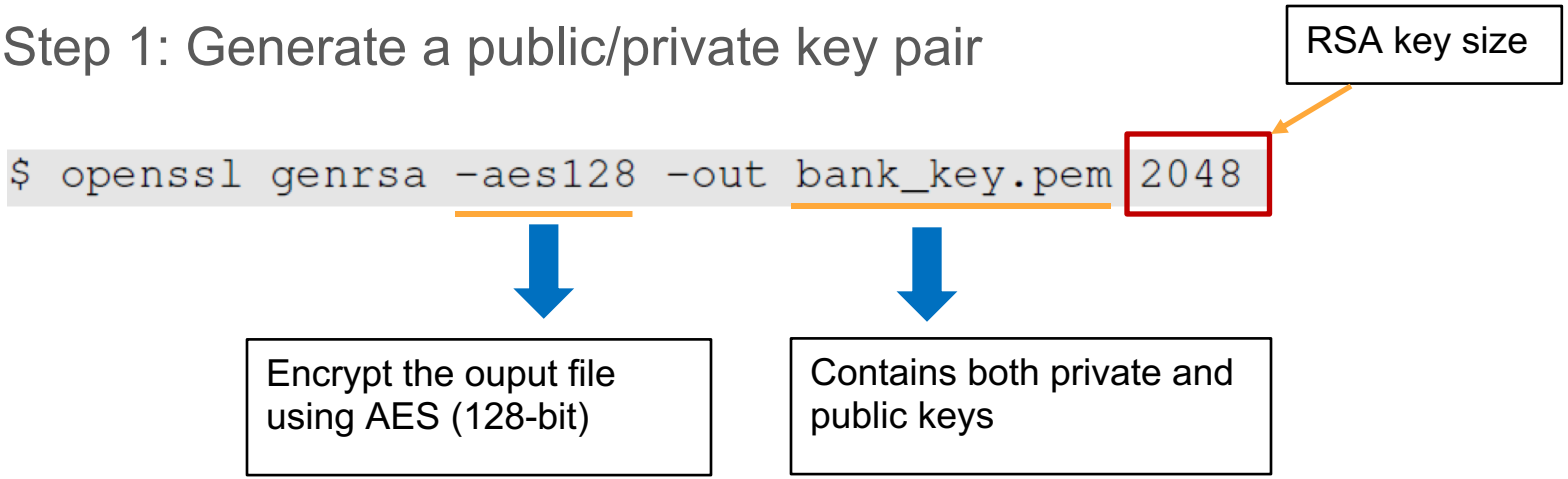
Discussion Question

- Question: If the ModelCA's certificate is self-signed, how do we verify it?

- Answer: There is no way to verify it. We just make sure that the certificate is obtained in a trusted way
  - Come with the operating system (if we trust OS, we trust the cert.)
  - Come with the software (if we trust the software, we trust the cert.)
  - Manually added (if we trust our own decision, we trust the cert.)
  - Sent to us by somebody whom we don't trust (don't trust the cert.)

Get a Certificate from CA: Step 1

- Step 1: Generate a public/private key pair



Get a Certificate from CA: Step 2

- Step 2: Generate a certificate signing request (CSR); identity information needs to be provided

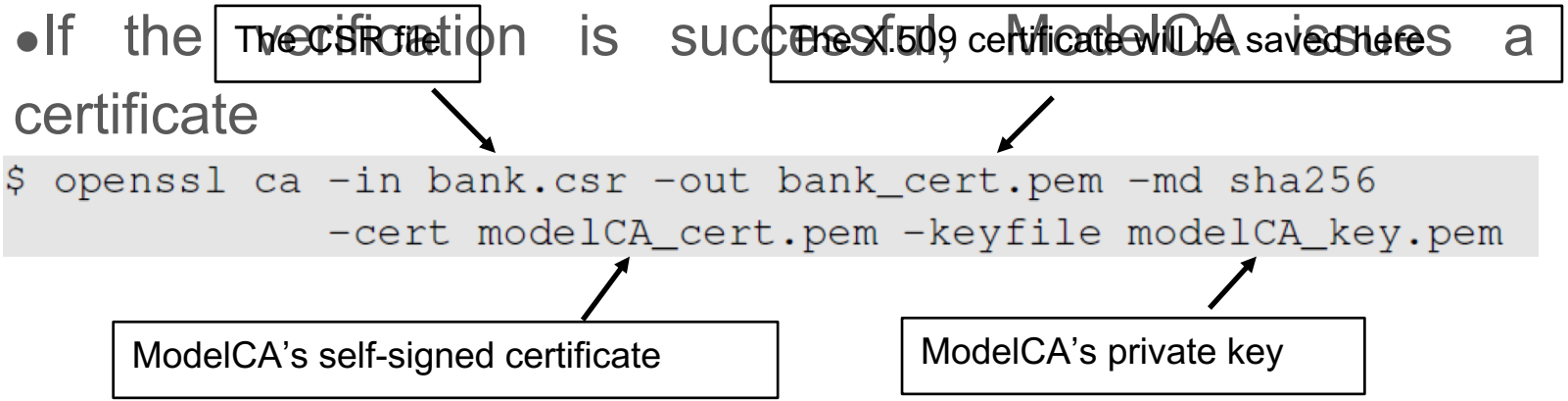
```
$ openssl req -new -key bank_key.pem -out bank.csr -sha256
```

```
$ openssl req -in bank.csr -text -noout
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=US, ST=New York, L=Syracuse, O=Example Inc,
                 CN=example.com/emailAddress=email@example.com
```

CA will verify this subject information

CA: Issuing X.509 Certificate

- We (the bank) need to send the CSR file to ModelCA.
- ModelCA will verify that we are the actual owner of (or can represent) the identity specified in the CSR file.
- If the verification is successful, ModelCA issues a certificate



Deploying Public Key Certificate in Web Server

- We will first use openssl's built-in server to set up an HTTPS web serve

```
$ cp bank.key bank.pem
$ cat bank.crt >> bank.pem
$ openssl s_server -cert bank.pem -accept 4433 -www
```

- A

```
example.com:4433 uses an invalid security certificate.

The certificate is not trusted because no issuer chain was provided.
The certificate is only valid for example.com
(Error code: sec_error_unknown_issuer)
```

we

g

Answer to the Question in the Previous Slide

- Firefox needs to use ModelCA's public key to verify the certificate
- Firefox does not have ModelCA's public key certificate
- We can manually add ModelCA's certificate to Firefox

Goto `Edit -> Preference -> Advanced -> View Certificates`

Import `ModelCA_cert.pem`

Apache Setup for HTTPS

- We add the following VirtualHost entry to the Apache configuration file:

```
<VirtualHost *:443>
    ServerName example.com
    DocumentRoot /var/www/Example
    DirectoryIndex index.html

    SSLEngine On
    SSLCertificateFile      /etc/apache2/ssl/bank_cert.pem   ①
    SSLCertificateKeyFile   /etc/apache2/ssl/bank_key.pem    ②
</VirtualHost>
```

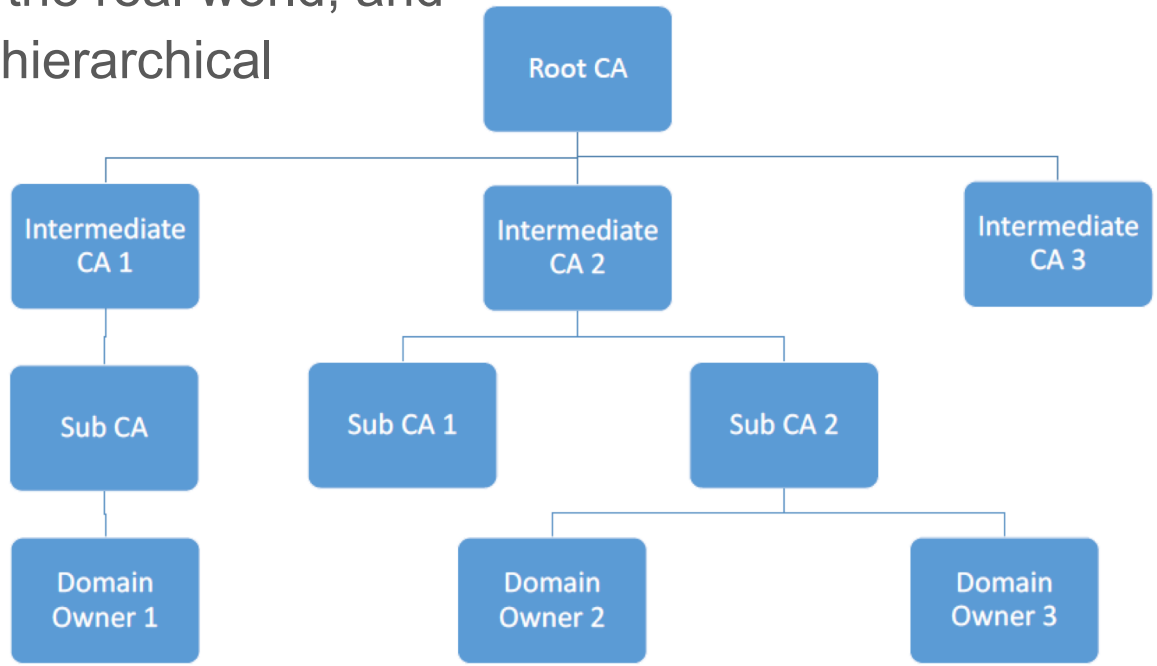
The server's certificate

The server's private key

Note: Apache configuration file is located at
`/etc/apache2/sites-available/default`

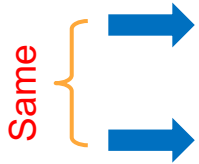
Root and Intermediate Certificate Authorities

There are many CAs in the real world, and they are organized in a hierarchical structure.



Root CAs and Self-Signed Certificate

- A root CA's public key is also stored in an X.509 certificate. It is self-signed.
- Self-signed: the entries for the issuer and the subject are identical.




```
Issuer: C=US, O=VeriSign, Inc., OU=VeriSign Trust Network,
    OU=(c) 2006 VeriSign, Inc. - For authorized use only,
    CN=VeriSign Class 3 Public Primary Certification Authority - G5
Subject: C=US, O=VeriSign, Inc., OU=VeriSign Trust Network,
    OU=(c) 2006 VeriSign, Inc. - For authorized use only,
    CN=VeriSign Class 3 Public Primary Certification Authority - G5
```

- How can they be trusted?
  - Public keys of root CAs are pre-installed in the OS, browsers and other software

Intermediate CAs and Chain of Trust

```
$ openssl s_client -showcerts -connect www.paypal.com:443

Certificate chain
 0 s: ... /CN=www.paypal.com
   i: ... /CN=Symantec Class 3 EV SSL CA - G3
-----BEGIN CERTIFICATE-----
MIIHWTCCBkGgAwIBAgIQLNGVEFQ30N5KOSAFavbCfzANBgkqhkiG9w0BAQsFADB3
...
-----END CERTIFICATE-----
 1 s: ... /CN=Symantec Class 3 EV SSL CA - G3
   i: ... /CN=VeriSign Class 3 Public Primary Certification
              Authority - G5
-----BEGIN CERTIFICATE-----
MIIFKzCCBBOgAwIBAgIQfuFKb2/v8tN/P61lTTIatDANBgkqhkiG9w0BAQsFADCB
...
-----END CERTIFICATE-----
```

Paypal's certificate

Intermediate CA's certificate

B

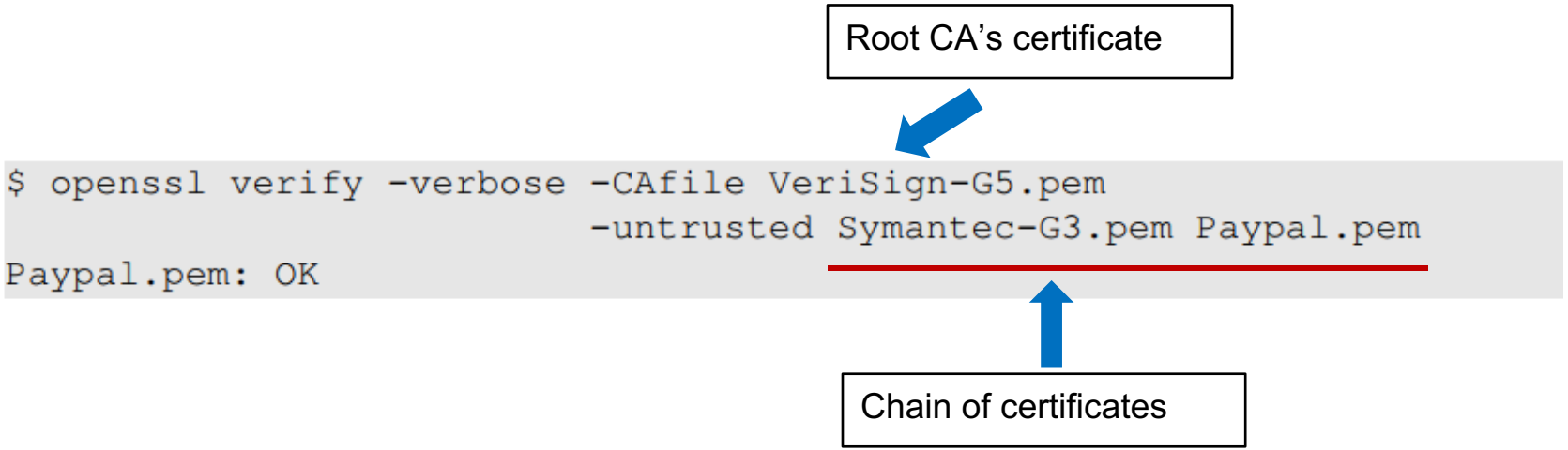
A

A is used to verify B

Something else is need to verify A (certificate from another intermediate CA or root CA)

Manually Verifying a Certificate Chain

- Paypal.pem: Save Paypal's certificate to a file called
- Symatec-g3.pem: Save certificate from "Symantec Class 3 EV SSL CA – G3"
- VeriSign-G5.pem: Save the VeriSign-G5's certificate from the browser



Creating Certificates for Intermediate CA

- When generating a certificate for an intermediate CA, we need to do something special:

```
$ openssl ca -in modelIntCA.csr -out modelIntCA_cert.pem -md sha256
             -cert modelCA_cert.pem -keyfile modelCA_key.pem
        -extensions v3_ca
```

```
X509v3 extensions:
  X509v3 Basic Constraints:
      CA:TRUE
```

- The extension field of the certificate will look as follows:

TRUE means the certificate can be used verify other certificates, i.e, the owner is a CA. For non-CA certificates, this field is FALSE.

Apache Setup

- A server has a responsibility to send out all the intermediate CA's certificates needed for verifying its own certificate.
- In Apache, all certificates including those from Intermediate C[illegible]ctive.

```
<VirtualHost *:443>
    ServerName example.com
    DocumentRoot /var/www/Example
    DirectoryIndex index.html

    SSLEngine On
    SSLCertificateFile      /etc/apache2/ssl/bank_cert2.pem
    SSLCertificateKeyFile   /etc/apache2/ssl/bank_key.pem
    SSLCertificateChainFile /etc/apache2/ssl/modelIntCA_cert.pem
</VirtualHost>
```

Trusted CAs in the Real World

- Not all of the trusted CAs are present in all browsers.
- According to W3Techs in April 2017, Comodo takes most of the market share followed by IdenTrust, Symantec Group, GoDaddy Group, GlobalSign and DigiCert.
- The list of trusted CAs supported by browser can be found:
    - **For the Chrome browser:**
        - Settings -> Show advanced settings -> Manage Certificates
    - **For the Firefox browser:**
        - Edit -> Preferences -> Advanced -> Certificates -> View Certificates -> Certificate Manager -> Authorities

How PKI Defeats the MITM Attack

- Assume that Alice wants to visit `https://example.com`
- When the server sends its public key to Alice, an attacker intercepts the communication. The attacker can do the following things:
  - Attacker forwards the authentic certificate from example.com
  - Attacker creates a fake certificate
  - Attacker sends his/her own certificate to Alice

Attacker Forwards the Authentic Certificate

- Attacker (Mike) forwards the authentic certificate
- Alice sends to the server a secret, encrypted using the public key.
- The secret is used for establishing an encrypted channel between Alice and server
- Mike doesn’t know the corresponding private key, so he cannot find the secret.
- Mike can’t do much to the communication, except for DoS.
- **MITM attack fails**.

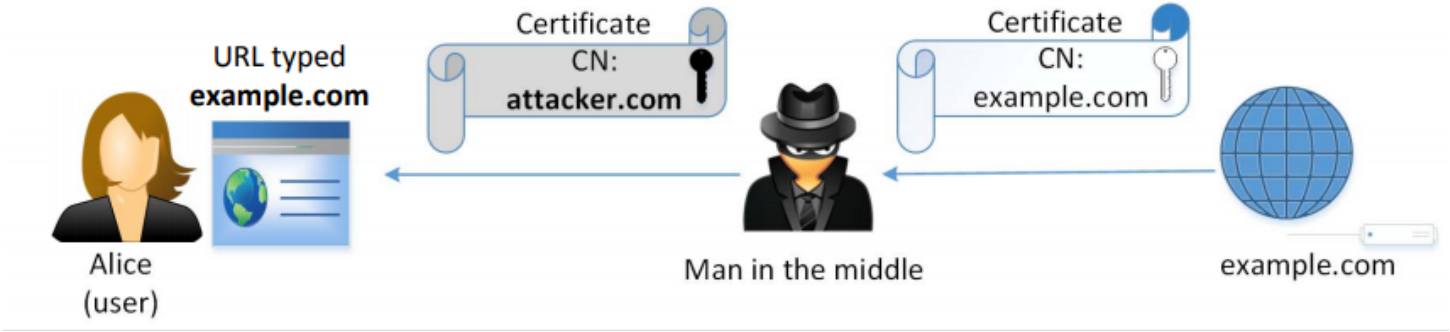
Attacker Creates a Fake Certificate

- Attacker (Mike) creates fraudulent certificate for the example.com domain.
- Mike replaces the server's public with his own public key.
- Trusted CAs will not sign Mike's certificate request as he does not own example.com.
- Mike can sign the fraudulent certificate by himself and create a self-signed certificate.
- Alice's browser will not find any trusted certificate to verify the received certificate and will give the following warning:

```
example.com uses an invalid security certificate.
The certificate is not trusted because it is self-signed.
```

- **MITM attack fails** if the user decide to terminate the connection

Attacker Sends His/Her Own Certificate



- Attacker's certificate is valid.
- Browser checks if the identity specified in the subject field of the certificate matches the Alice's intent.
  - There is a mismatch: attacker.com ≠ example.com
- Browser terminates handshake protocol: **MITM fails**

Emulating an MITM Attack

- DNS Attack is a typical approach to achieve MITM
  - We emulate an DNS attack by manually changing the /etc/hosts file on the user's machine to map example.com to the IP address of the attacker's machine.
- On attacker's machine we host a website for example.com.
  - We use the attacker's X.509 certificate to set up the server

```
example.com uses an invalid security certificate.
The certificate is only valid for attacker32.com
(Error code: ssl_error_bad_cert_domain)
```

attacker32.com

When we visit example.com, we get an error message:

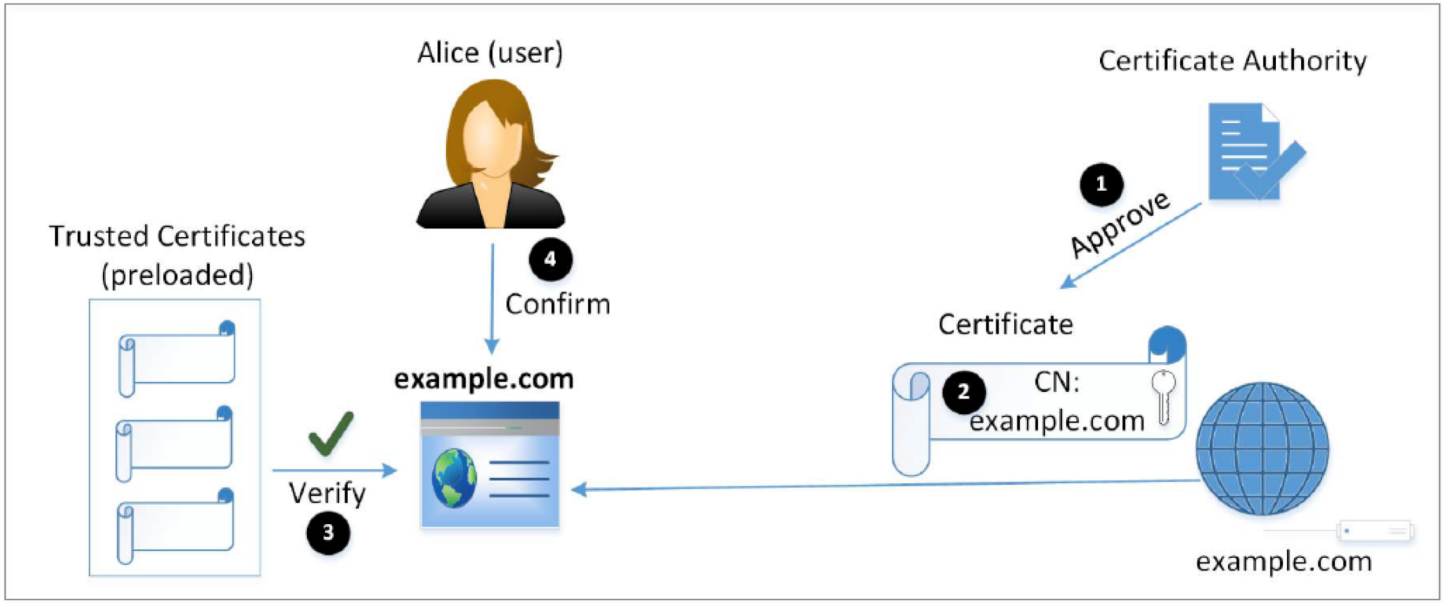
The Importance of Verifying Common Name

- During TLS/SSL handshake browsers conduct two important validations
  1) Checks whether the received certificate is valid or not.
  2) Verifies whether the subject (Common Names) in the certificate is the same as the hostname of the server.
- Not verifying the common name is a common mistake in software

The Man-In-The-Middle Proxy

- Proxy creates a self-signed CA certificate, which is installed on the user's browser
- The routing on the user machine is configured; all outgoing HTTPS traffic is directed towards the proxy machine
- When user tries to visit an HTTPS site:
  - Proxy intercepts communication
  - Creates a fake certificate
  - Browser already has the proxy's certificate in its trusted list to be able to verify all the fake certificates
  - Proxy becomes MITM

Attacks Surfaces on PKI



Attack on CA's Verification Process

- CA's job has two parts:
  - Verify the relationship between certificate applicant and the subject information inside the certificate
  - Put a digital signature on the certificate

- **Case study: Comodo Breach [March 2011]**
  - Popular root CA.
  - The approval process in Southern Europe was compromised.
  - Nine certificates were issued to seven domains and hence the attacker could provide false attestation.
  - One of the affected domain (a key domain for the Firefox browser): `addons.mozilla.org`

Attack on CA’s Signing Process

- If the CA’s private key is compromised, attackers can sign a certificate with any arbitrary data in the subject field.
- **Case Study: the DigiNotar Breach [June-July 2011]**
  - A top commercial CA
  - Attacker got DigiNotar’s private key
  - 531 rogue certificates were issued.
  - Traffic intended for Google subdomains was intercepted: MITM attack.
- How CAs Protect Their Private Key
  - Hardware Security Model (HSM)

Attacks on Algorithms

- Digital Certificates depend on two types of algorithms
  - one-way hash function and digital signature

- **Case Study: the Collision-Resistant Property of One-Way Hash**
  - At CRYPTO2004, Xiaoyun Wang demonstrated collision attack against MD5.
  - In February 2017, Google Research announced SHAttered attack
    - Attack broke the collision-resistant property of SHA-1
    - Two different PDF files with the same SHA-1 has was created.

- Countermeasures: use stronger algorithm, e.g. SHA256.

Attacks on User Confirmation

- After verifying the certificate from the server, client software is sure that the certificate is valid and authentic
- In addition, the software needs to confirm that the server is what the user intends to interact with.
- Confirmation involves two pieces of information
  - Information provided or approved by user
  - The common name field inside the server's certificate
  - Some software does not compare these two pieces of information: security flaw

Attacks on Confirmation: Case Study

Phishing Attack on Common Name with Unicode

- Zheng found out several browsers do not display the domain name correctly if name contains Unicode.
- `xn–80ak6aa92e.com` is encoded using Cyrillic characters. But domain name displayed by browser likes like `apple.com`
- Attack:
  - Get a certificate for `xn–80ak6aa92e.com`
  - Get user to visit `xn–80ak6aa92e.com`, so the common name is matched
  - User's browser shows that the website is apple.com. **User can be fooled**.
- Had the browser told the user that the actual domain is not the real apple.com, the user would stop.

Types of Digital Certificate

- Domain Validated Certificates (DV)
- Organizational Validated Certificates (OV)
- Extended Validated Certificates (EV)

Domain Validated Certificates (DV)

- Most popular type of certificate.
- The CA verifies the domain records to check if the domain belongs to applicant.
- Domain Control Validation (DCV) is performed on domain name in the certificate request.
- DCV uses information in the WHOIS database
- DCV is conducted via
    - Email
    - HTTP
    - DNS

Organizational Validated Certificates (OV)

- Not very popular type of certificate.
- CAs verify the following before issuing OV certificates:
  - Domain control validation.
  - Applicant’s identity and address.
  - Applicant’s link to organization.
  - Organization’s address.
  - Organization’s WHOIS record.
  - Callback on organization’s verified telephone number.

Extended Validated Certificates (EV)

- CAs issuing EV certificates require documents that are legally signed from registration authorities.
- EV CA validate the following information:
  - Domain control validation.
  - Verify the identity, authority, signature and link of the individual.
  - Verify the organization's physical address and telephone number.
  - Verify the operational existence.
  - Verify the legal and proper standings of the organization.
- EV certificate, hence, costs higher but is trustworthy.

How Browsers Display Certificate Types

Chrome browser

Cannot be verified

DV/OV Certificate

EV Certificate

Firefox browser

Cannot be verified

DV/OV Certificate

EV Certificate

Summary

- MITM attacks on public key cryptography
- Public-Key Infrastructure
- X.509 digital certificate
- Certificate Authority and how CA signs certificate
- How PKI defeats MITM attacks
- Attacks on PKI
- Different types of digital certificate