Lecture 25: Isolation

Stephen Huang

Content

1. Principles of Secure Design
2. Firewalls
3. Sandboxing
4. Web Isolation

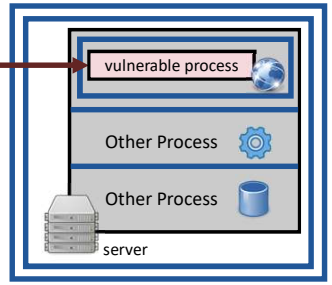
1. Principles of Secure Design

- Methodology
  - As a basic start, establish secure defaults, minimize the attack surface area, and fail securely to those well-defined and understood defaults.
  - Two processes: Product Inception and Product Design.
- Security Principles
  - The principle of Least Privilege and Separation of Duties.
  - The principle of Defense-in-Depth.
  - The principle of Zero Trust
  - The principle of Security-in-the-Open

Secure Product Design Cheat Sheet, https://cheatsheetseries.owasp.org/cheatsheets/Secure_Product_Design_Cheat_Sheet.htm

Principles of Secure Design

- Defense-in-depth: multiple layers of security
  - The attacker has to circumvent all of them to compromise its target.
  - *examples*: multi-factor user authentication, firewalls, etc.



- Compartmentalization
  - divide the system into compartments and isolate them from each other → limit the impact of a compromise
  - **principle of least privilege**: each module should have only the minimum set of privileges needed to serve its purpose

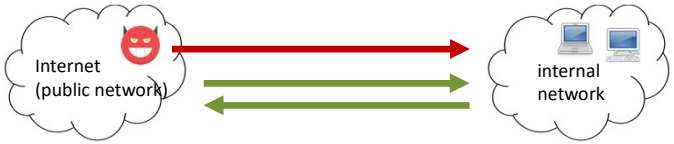
2. Firewalls

- A Firewall is a network security device that monitors and filters incoming and outgoing network traffic based on an organization's established security policies.

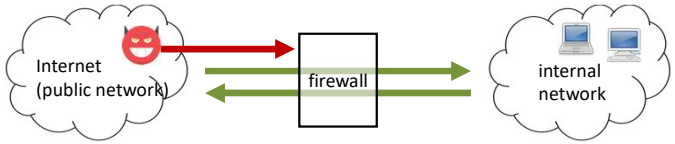
Motivation for Firewalls

- A system can consist of hundreds or thousands of computing devices (both servers and clients)
  - many of them may have open ports that are accessible from the Internet
  - various operating systems running various services, implemented by various software, …
  - various software versions, configurations, …
- Challenge
  - keeping everything patched and properly configured is very expensive (or practically impossible)
  - on the other hand, an attacker may succeed by finding just one vulnerability (which it may easily find by scanning the network)

Basic Layout



Basic Layout



decision whether to allow or deny some traffic is based on a set of rules (*e.g.*, allow connections to the webserver, but deny connections to other machines)

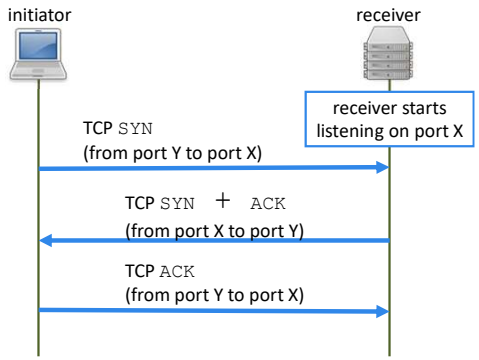
Transport Layer Protocols: TCP & UDP

- TCP or UDP port: communication endpoint
  - for each address and protocol (TCP or UDP), it is identified by a 16-bit number
  - every TCP and UDP packet has a source address and source port as well as a destination address and destination port
- Ports 0 … 1023: reserved for certain application-level protocols
  - on each computer, these ports are permanently assigned to the programs that implement the corresponding services
  - *examples:* HTTP → TCP 80, DNS → TCP / UDP 53, FTP control → TCP 21
- Ports 1024 … 65535: used by client applications
  - client picks an arbitrary port, and connects from this port to the specific port on the server
  - on Unix-like systems, unprivileged processes cannot listen on ports below 1024

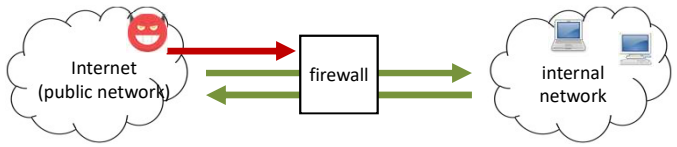
TCP Handshake

- TCP uses a three-way handshake to establish a connection

Each packet has zero or more flags: `SYN`, `ACK`, `FIN`, …



Basic Layout



decision whether to allow or deny some traffic is based on a set of rules
(*e.g.*, allow connections to the webserver, but deny connections to other machines)

Firewalls: Stateless vs. Stateful

Stateless
(Packet filtering)

- Apply rules to each incoming/outgoing packet individually
- Advantage: does not need to track state → simpler
- Disadvantage: limited functionality, must process rules for every packet

Stateful
(Session filtering)

- Apply rules only to the first few packets of each connection
- Advantage: advanced functionality, does not need to process every packet
- Disadvantage: must track the state of every connection

Illustration of Basic Packet Filtering

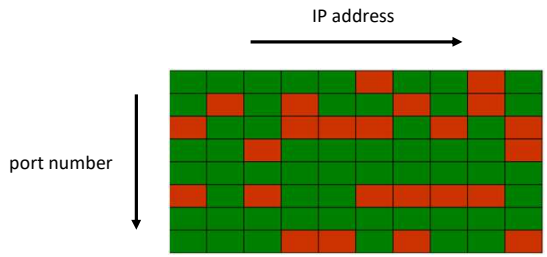
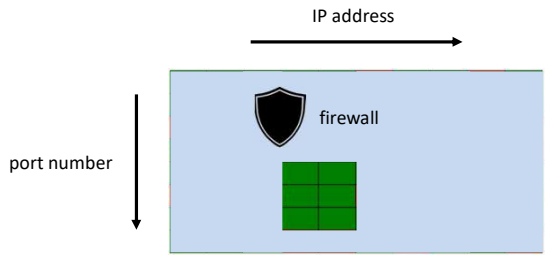


Illustration of Basic Packet Filtering



Basic Packet Filtering

- Based on protocol header information
  - network layer: source and destination IP address, higher level protocol (*e.g.*, TCP, UDP, ICMP)
  - transport layer: TCP or UDP ports, TCP flags (SYN, ACK, FIN, …)
  - ICMP message types (*e.g.*, ECHO Request)
- *Example ruleset:*

`from Internet to 129.7.97.54 ∧ TCP port = 80 → ALLOW`

- allow connections from the Internet to our webserver (i.e., port 80 on 129.7.97.54)

`from Internet to LAN ∧ TCP SYN=true ∧ TCP ACK=false → DENY`

- deny connections from the Internet to computers on the internal network

`otherwise → ALLOW`

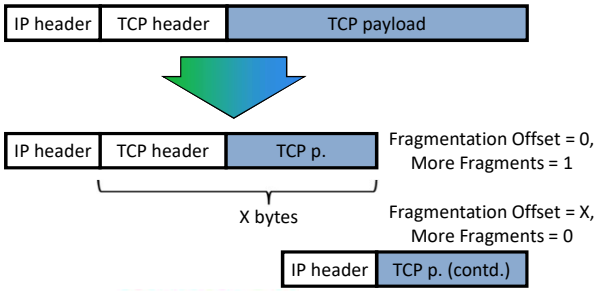
- allow everything else (note that this will allow computers on the Internet to accept connections from the internal network)

Rulesets

- Multiple matching rules
  - first match → first matching rule is applied
  - last match → last matching rule is applied
- Actions
  - usually accept/drop (i.e., allow/deny)
  - can also include logging or alarms
- Ruleset needs to be built for the specific system
  - "off-the-shelf" configuration might not fit the system
- Best practice
  - last (or first) rule should always be a "block everything," so only explicitly allowed traffic will pass through
  - in other words, "whitelist" instead of "blacklist"

Stateless Filtering Limitation

- IP fragmentation: breaking up a single IP packet into multiple packets
  - necessary because some network links have limited datagram size
- Normal fragmentation



IP Packet Fragmentation Attack

- *Example firewall rules:*
  - allow connections on TCP port 25 (SMTP)
  - deny connection on TCP port 23 (Telnet)
- First packet
  - Fragmentation Offset = 0
  - More Fragment = 1
  - TCP header: destination port = 25 → allow
- Second packet
  - Fragmentation Offset = 2 (second packet overwrites all but the first 2 bytes of the previous packet)
  - More Fragment = 0
  - TCP header fragment: destination port = 23
- Firewall allows the second packet because it is just a fragment (not a full header)
- IP packet reassembled at host and received at port 23

Stateful Firewalls and Deep Packet Inspection

- Stateful firewalls
  - keep track of each active connection
  - each packet is evaluated based on the status of the connection
  - disadvantages:
    - computationally more demanding
    - complex → more error-prone implementation
- Deep packet inspection
  - looks into the internals of a packet to check application content or context
  - example: block certain HTTP URLs or malicious executable code
  - may lead to privacy issues

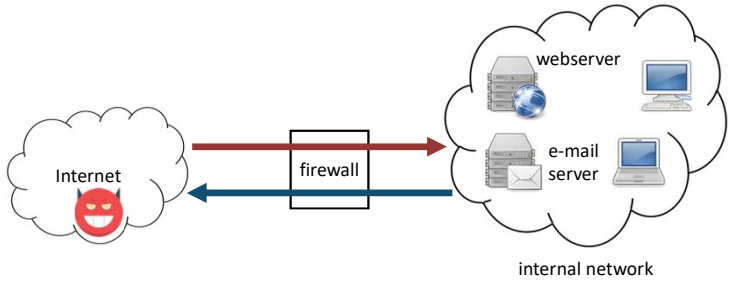
Application-Layer Firewalls

- Basic packet inspection considers only the first four networking layers (up to the transport layer).
- It operates at the OSI model's application layer (or Layer 7).
- While traditional firewalls focus on packet filtering and IP addresses, application-layer firewalls dive deeper, inspecting the data's content to make more informed security decisions.
  - It's like a meticulous security guard who checks not just your ID, but also the contents of your bag before letting you inside a building.

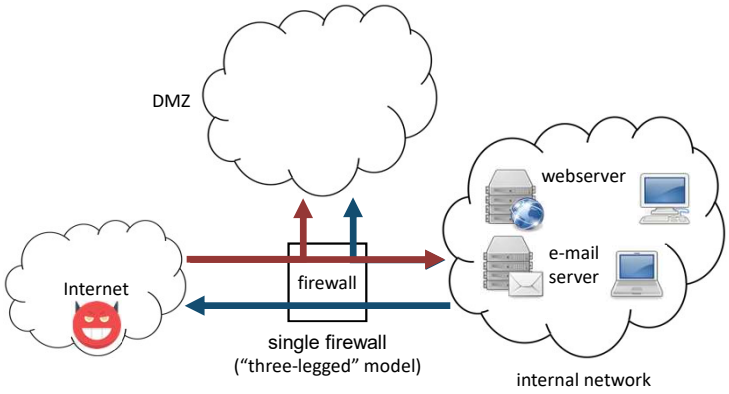
Application-Layer Firewalls

- Application-layer firewall
  - "understands" certain application-level protocols (e.g., FTP, DNS, HTTP)
  - rules can be defined in terms of these protocols
    - e.g., limit HTTP requests to certain paths or limit FTP to certain commands
- Proxying firewall
  - application-layer firewalls are sometimes implemented as proxies
  - client TCP connection is received by the proxy, which then connects to the actual server
  - proxy can inspect and forward traffic

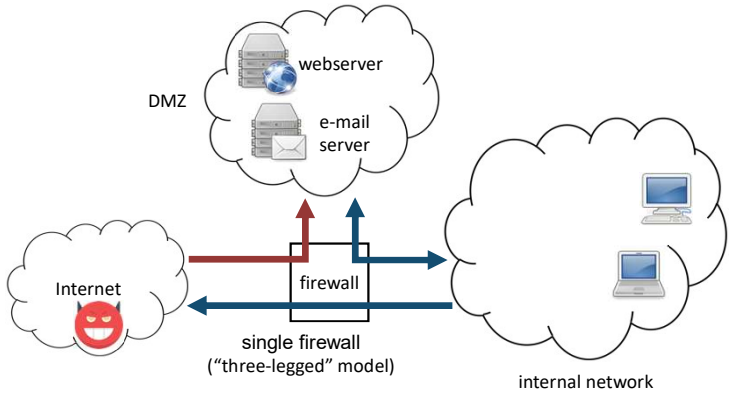
Firewall



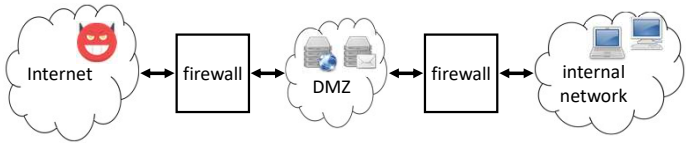
Demilitarized Zone (DMZ)



Demilitarized Zone (DMZ)

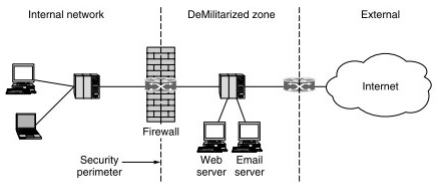


DMZ with Dual Firewalls



- More secure
  - The attacker needs to compromise/bypass two firewalls.
  - Misconfiguration has a lower impact.

Various Configurations



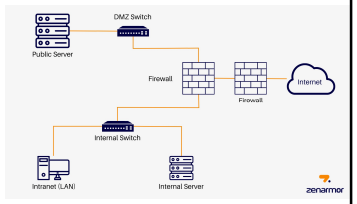
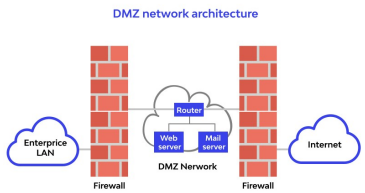
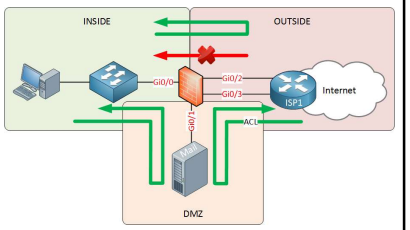


Figure 8-29. A firewall protecting an internal network.





Firewall Implementations

- Many operating systems come with built-in firewalls, *e.g.*,
  - Windows Firewall
  - Linux kernel: Netfilter framework
    - `iptables`: command-line tool for configuration
- Many routers have basic firewall functionality
- Dedicated hardware devices
  - special hardware and software for filtering
  - advantage: performance
  - disadvantage: expensive and usually harder to manage

Firewall Limitations

- Firewalls cannot protect against
  - attacks that bypass them (*e.g.*, USB drive with malware)
  - internal-threats (*e.g.*, disgruntled employee)
- A firewall can become a single point of failure
  - just like any other machine, a firewall can have vulnerabilities
  - if the security of an entire network depends on a firewall, then compromising the firewall can have devastating effects

3. Sandboxing

Sandbox

- Sandbox: security mechanism for separating a running program (or one part of a running program) from the rest of the system
- Typically used to run untested or untrusted code (possibly from untrusted sources)
  - potentially vulnerable applications that are exposed to untrusted data (e.g., PDF viewer, e-mail client, web browser)
  - client-side scripts (~JavaScript) on webpages, macros, etc.
  - plug-ins, extensions, smartphone applications, etc.
- Sandboxed code has limited access to system resources (*e.g.*, files, memory, network) → reduces the impact of security vulnerabilities in sandboxed software

`chroot` Jail

```
int chroot(const char *path)
```

- Available on all Unix operating systems
- Changes the root directory from `/` to the given `path` for the process
- Can be performed only with root user privileges
- After `chroot`, nothing outside the new root directory is available
  - system libraries and commands (*e.g.*, `libc`, `bash`) cannot be used, unless a copy was made available in the new root
- By definition, it should be undoable
  - however, a root user can get out
    → drop root privileges before executing the untrusted code

Using `chroot`

- Development and testing
  - set up a complete environment in a directory and chroot into it before running the software that is under development or testing
- Security
  - if an attacker compromises a process running in a "chroot jail," then it can access and modify only the files inside the jail
  - *example*:
    - HTTP, e-mail, and FTP servers (*e.g.*, Postfix and OpenSSH SFTP)
    - before handling a client, `chroot` into a directory and relinquish root privileges
- Disadvantages
  - all or nothing access to parts of a file system

Linux Secure Computing Mode

`int seccomp(SECCOMP_SET_MODE_STRICT, 0, NULL)`

- Sandboxing tool in the Linux kernel
  - introduced in kernel version 2.6.12
- Once a process enters `seccomp` mode, it cannot make any system calls, except for
  - `exit(…), sigreturn(…)`
  - `read(int filedescriptor, …)`
  - `write(int filedescriptor, …)`

  → read and write files that were opened before entering `seccomp` mode
- Typical usage: open necessary files and network connections, and enter `seccomp` mode

seccomp-bpf

`int seccomp(SECCOMP_SET_MODE_FILTER, 0, &filter)`

- Extension to secure computing mode that enables more fine-grained filtering of system calls
- Berkeley Packet Filter
  - enables user-space processes to filter network packets using filters
  - in `seccomp`, it is used to filter system calls
  - `&filter` points to a struct `sock_fprog`, which contains the filter
- Used by, for example,
  - Google Chrome (sandbox for Adobe Flash and renderers)
  - Firefox (sandbox for child processes and plugins)
  - vsftpd (default FTP server for many Linux distributions)

Isolation Based on Unix Access Control

- *Reminder:* traditional Unix access control
  - each user has a user ID and a set of group memberships
  - when a user starts a process, the process inherits the user's user ID and set of group memberships
  - each file has
    - owner (user ID) and group (group ID)
    - read, write, and execution rights for owner, group, and others
- Sandboxing: run untrusted code as an unprivileged user
  - for example, use `setuid` to run an executable as a dummy user
  - dummy user's access can be restricted using traditional Unix access control
  - using Linux namespaces, the running process can be further separated

Android Application Sandbox

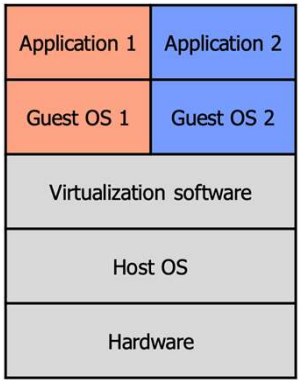
- Smartphones allow users to easily download and install applications → OS must prevent applications (*e.g.*, trojans) from doing harm
- On Android, each application has a unique Unix user ID and group ID
- Application data is stored in `/data/data/<app-name>`
  - read-writable only with the corresponding user / group IDs
- Applications run in separate processes with their own user IDs and group IDs
- Application permissions are implemented using Unix groups, *e.g.*,
  - `WRITE_EXTERNAL_STORAGE` permission: `sdcard_rw` group (can read/write `/mnt/sdcard`)
  - `INTERNET` permission: `inet` group (can create IP sockets)

Other Sandbox Approaches

- Linux Security Modules
  - framework that allows extending the traditional Unix access control to various computer-security models
  - allows a security module to intercept system calls that would result in accessing important kernel objects
  - *examples:* SELinux, AppArmor
- `ptrace`
  - system call available on most Unix systems
  - enables a process to control another process (*e.g.*, manipulate file descriptors and memory, install breakpoints)
  - commonly used by debuggers (*e.g.*, `gdb`)
  - can be used to implement a sandbox

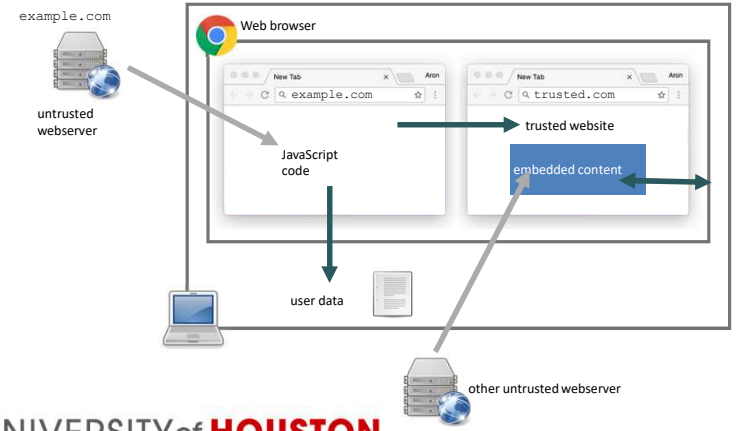
Virtual Machines

- Virtual machines can also be used for sandboxing
- Widely used in cloud computing
  - *Microsoft Azure*
  - *HP Public Cloud*
  - *IBM Cloud Services*
  - …



4. Web Isolation

Code Isolation in Web Browsers



Embedding Example

- HTML page source:

```
<iframe style="width:
122px; height: 20px;
border: none;"
src="https://www.facebook.c
om/v2.3/plugins/like.php?ap
p_id=24725…"></iframe>
```

HTML5 `iframe` Sandbox

- Motivation
  - lot of websites embed content from other domains (*e.g.*, advertisements, Facebook modules)
  - third-party widgets can run Javascript, which may open pop-ups or navigate to another page

Sandboxed Embedding

- Without sandboxing:

```
<iframe src="untrusted.html"></iframe>
```

- With sandboxing:

```
<iframe src="untrusted.html" sandbox></iframe>
```

  - disables plugins
  - blocks script execution
  - blocks form submission
  - treats content as if it was from a globally unique origin
  - blocks navigating the top-level window or other frames on the page (excluding child frames of the sandboxed content)
  - blocks popup windows

Refining the Sandbox

- Manually allow some features

```
<iframe src="untrusted.html"
    sandbox="allow-forms"></iframe>
```

  - `allow-forms`: allows form submission
  - `allow-popups`: allows displaying pop-ups
  - `allow-same-origin`: treats content as being from the same origin
  - `allow-scripts`: allows script execution
  - `allow-top-navigation`: allows the iframe content to navigate its top-level browsing context

Next Topic

- Isolation
- Denial of Service
- Conclusion