CIS 433/533 - Computer and Network Security
Firewalls and Web

Professor Kevin Butler
Winter 2011
Firewalls

- A firewall ... is a physical barrier inside a building or vehicle, designed to limit the spread of fire, heat and structural collapse.
Filtering: Firewalls

• Filtering traffic based on *policy*
  ‣ Policy determines what is acceptable traffic
  ‣ Access control over traffic
  ‣ Accept or deny

• May perform other duties
  ‣ Logging (forensics, SLA)
  ‣ Flagging (intrusion detection)
  ‣ QoS (differentiated services)
### IP Firewall Policy

- Specifies what traffic is (not) allowed
  - Maps attributes to address and ports
  - Example: HTTP should be allowed to any external host, but inbound only to web-server

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Flags</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>1.1.1.1</td>
<td>80</td>
<td>TCP</td>
</tr>
<tr>
<td>1.1.1.*</td>
<td>*</td>
<td>*</td>
<td>80</td>
<td>TCP</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>80</td>
<td>TCP</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>TCP</td>
</tr>
</tbody>
</table>

* Denotes wildcard.
• **Blacklisting** - specifying specific connectivity that is explicitly disallowed
  ‣ E.g., prevent connections from badguys.com

• **Whitelisting** - specifying specific connectivity that explicitly allowed
  ‣ E.g., allow connections from goodguys.com

• These is useful for IP filtering, SPAM mitigation, …

• Q: What access control policies do these represent?
Stateful, Proxy, and Transparent

• Single packet contains insufficient data to make access control decision
  ‣ **Stateful**: allows historical context consideration
  ‣ Firewall collects data over time
    • e.g., TCP packet is part of established session

• Firewalls can affect network traffic
  ‣ **Transparent**: appear as a single router (network)
  ‣ **Proxy**: receives, interprets, and reinitiates communication (application)
    ‣ Transparent good for speed (routers), proxies good for complex state (applications)
DMZ (De-militarized Zone)

- Zone between LAN and Internet (*public facing*)
Practical Issues and Limitations

• Network layer firewalls are dominant
  ‣ DMZs allow multi-tiered fire-walling
  ‣ Tools are widely available and mature
  ‣ Personal firewalls gaining popularity

• Issues
  ‣ Network perimeters not quite as clear as before
    • E.g., telecommuters, VPNs, wireless, …
  ‣ Every access point must be protected
    • E.g., this is why war-dialing is effective
  ‣ Hard to debug, maintain consistency and correctness
  ‣ Often seen by non-security personnel as impediment
    • E.g., Just open port $X$ so I can use my wonder widget …

• SOAP - why is this protocol an issue?
Practical Firewall Implementations

• Primary task is to filter packets
  ‣ But systems and requirements are complex

• Consider
  ‣ All the protocols and services
  ‣ Stateless vs. stateful firewalls
  ‣ Network function: NAT, forwarding, etc.

• Practical implementation: Linux iptables
The iptables firewall looks in the firewall table to seek if the chain associated with the current hook matches a packet, and executes the target if it does.

- **Table**: allows policies to be cleanly separated by purpose (default: “-t filter”, also: “-t nat”, “-t mangle” and “-t raw”) Each table as a set of default chains.
- **Chain**: list of rules associated with the chain identifier, e.g., hook name (INPUT, OUTPUT, etc)
- **Match**: when all a rule’s field match the packet
- **Target**: operation to execute on a packet given a match
Targets

- Define what to do with the packet at this time
  - ACCEPT/DROP
  - QUEUE for user-space application
  - LOG any packet that matches
  - REJECT drops and returns error packet
  - RETURN enables packet to return to previous chain
  - `<user-specified>` passes packet to that chain

```bash
## Create chain which blocks new connections, except if coming from inside.
# iptables -N block
# iptables -A block -m state --state ESTABLISHED,RELATED -j ACCEPT
# iptables -A block -m state --state NEW -i ! ppp0 -j ACCEPT
# iptables -A block -j DROP

## Jump to that chain from INPUT and FORWARD chains.
# iptables -A INPUT -j block
# iptables -A FORWARD -j block
```
Examples

iptables -A INPUT -s 200.200.200.2 -j ACCEPT
iptables -A INPUT -s 200.200.200.1 -j DROP
iptables -A INPUT -s 200.200.200.1 -p tcp -j DROP
iptables -A INPUT -s 200.200.200.1 -p tcp --dport telnet -j DROP
iptables -A INPUT -p tcp --destination-port telnet -i ppp0 -j DROP
Example: Gateway/DMZ Firewalls

• Assume you have two firewalls (FW1 and FW2), each with two ethernet interfaces (eth0 and eth1).

• FW1 protects the DMZ, and FW2 protects the LAN.

• Define an iptables policy for FW1 that
  ‣ Allows new Internet traffic to reach port 80 on 10.0.1.13
  ‣ Does not allow traffic to reach the LAN (10.0.2.0/24)

• Define an iptables policy for FW2 that
  ‣ Allows internal hosts to reach the webserver, but nothing else in the DMZ (10.0.1.0/24)
  ‣ Prevents DMZ hosts from initiating connections to LAN
Example: Gateway/DMZ Firewalls

• FW1 Policy

```sh
# iptables -F INPUT
# iptables -F OUTPUT
# iptables -F FORWARD
# iptables -P INPUT ACCEPT
# iptables -P OUTPUT ACCEPT
# iptables -P FORWARD DROP

# iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
# iptables -A FORWARD -i eth0 -o eth1 -m state --state NEW -d 10.0.1.13 --dport 80 -j ACCEPT
# iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT
```

• FW2 Policy

```sh
# iptables -F INPUT
# iptables -F OUTPUT
# iptables -F FORWARD
# iptables -P INPUT ACCEPT
# iptables -P OUTPUT ACCEPT
# iptables -P FORWARD DROP

# iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
# iptables -A FORWARD -i eth1 -o eth0 -m state --state NEW -d 10.0.1.13 --dport 80 -j ACCEPT
# iptables -A FORWARD -i eth1 -o eth0 -d ! 10.0.1.0/24 -j ACCEPT
```
Example: Host Firewall

• Assume you have a host with one network interface (eth0). You are running SSH (port 22) and want to allow access by external hosts. You are also running Apache for Web development, and only want it to be accessed by other hosts on the LAN (10.0.2.0/24)

# iptables -F INPUT
# iptables -F OUTPUT
# iptables -F FORWARD
# iptables -P INPUT DROP
# iptables -P OUTPUT ACCEPT
# iptables -P FORWARD DROP

# iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
# iptables -A INPUT -i eth0 -m state --state NEW --dport 22 -j ACCEPT
# iptables -A INPUT -i eth0 -m state --state NEW --s 10.0.2.0/24 --dport 80 -j ACCEPT
Deep Packet Inspection

- **Deep packet inspection** looks into the internals of a pack to look for some application/content context
  - e.g., inspect HTTP for URLs that point to malicious websites
  - Can have serious privacy issues if done by, say COMCAST

To specify a match in `iptables`

- `iptables -A INPUT -p tcp -m string --algo bm --string ‘exe’`
  - matches to packet with content containing ‘exe’
- `iptables -A INPUT -p tcp -m length --length 10:100`
  - matches to packet with length between 10 and 100 bytes
  - Also, can specify ‘greater than 10’ by 10:
Network vs. Web Security
What is the web?

- A collection of application-layer services used to distribute content
  - Web content (HTML)
  - Multimedia
  - Email
  - Instant messaging
- Many applications
  - News outlets, entertainment, education, research and technology, ...
  - Commercial, consumer and B2B
Web security: the high bits

• The largest distributed system in existence
  ‣ threats are as diverse as applications and users
  ‣ But need to be thought out carefully …

• The stakeholders are …
  ‣ Consumers (users, businesses, agents, …)
  ‣ Providers (web-servers, IM services, …)

• Another way of seeing web security is
  ‣ Securing the web infrastructure such that the integrity, confidentiality, and availability of content and user information is maintained
Early Web Systems

- Early web systems provided a click-render-click cycle of acquiring web content.
  - Web content consisted of static content with little user interaction.
Web Transport Security: SSL

• Secure socket Layer (SSL/TLS)
• Used to authenticate servers
  ‣ Uses certificates, “root” CAs
• Can authenticate clients
• Inclusive security protocol
• Security at the socket layer
  ‣ Transport Layer Security (TLS)
  ‣ Provides
    • authentication
    • confidentiality
    • integrity
SSL Handshake

(1) Client Hello (algorithms, …)
(2) Server Hello (alg. selection, …)
(3) Server Certificate
(4) ClientKeyRequest
(5) ChangeCipherSuite
(6) ChangeCipherSuite
(7) Finished
(8) Finished
Participants: Alice/A (client) and Bob/B (server)

Crypto Elements: Random R, Certificate C, $k_i^+$ Public Key (of i)

Crypto Functions: Hash function $H(x)$, Encryption $E(k, d)$, Decryption $D(k, d)$, Keyed MAC $HMAC(k, d)$

1. Alice $\rightarrow$ Bob $R_A$

2. Bob $\rightarrow$ Alice $R_B, C_B$
   - Alice: pick pre-master secret $S$
   - Alice: calculate master secret $K = H(S, R_A, R_B)$

3. Alice $\rightarrow$ Bob $E(k_B^+, S), HMAC(K', CLNT' + [\#1, \#2])$
   - Bob: recover pre-master secret $S = D(k_B^-, E(k_B^+, S))$
   - Bob: calculate master secret $K = H(S, R_A, R_B)$

4. Bob $\rightarrow$ Alice $HMAC(K', SRV R' + [\#1, \#2])$

Note: Alice and Bob: IV Keys, Encryption Keys, and Integrity Keys 6 keys, where each key $k_i = g_i(K, R_A, R_B)$, and $g_i$ is key generator function.
SSL Tradeoffs

- **Pros**
  - Server authentication*
  - GUI clues for users
  - Built into every browser
  - Easy to configure on the server
  - Protocol has been analyzed like crazy

- **Cons**
  - Users don’t check certificates
  - Too easy to obtain certificates
  - Too many roots in the browsers
  - Some settings are terrible
Dynamic Content: CGI

- Common Gateway Interface (CGI)
  - Generic way to call external applications on the server
  - Passes URL to external program (e.g., form)
  - Result is captured and return to requestor

- Historically
  - “shell” scripts used to generate content
    - Very, very dangerous

- NOTE: server extensions are no better (e.g., servlets)
DC: Embedded Scripting

- Program placed directly in content, run on server upon request and output returned in content
  - MS active server pages (ASP)
  - PHP
  - mod_perl
  - server-side JavaScript
  - python, ...

- Nice at generating output
  - Dangerous if tied to user input
Applications/Plugins

• A *plugin* is a simply a program used by a browser to process content
  ‣ MIME type maps content to plugin
  ‣ Like any old application (e.g., RealAudio)
  ‣ Newer browsers have autoinstall features

• A kind of plug-in …
  ‣ (1997) David.exe
  ‣ “Free pornography …”

• Moral: beware of plugins
JavaScript

• Scripting Language used to improve the quality/experience
  ‣ Create dialogs, forms, graphs, …
  ‣ Built upon API functions (lots of different flavors)
  ‣ No ability to read local files, open connections …

• Security: No ability to read local files, open connections, but …
  ‣ DOS – the “infinite popup” script
    • Often could not “break out” with restarting computer
  ‣ Spoofing – easy to create “password” dialogs
Malicious content injection

• Currently, two central infection vectors

1. Website compromise (and insert IFRAMEs)

2. Advertising: the abuse of Ad syndication (malverts)
Malicious IFrame(s)

• An IFRAME is a HTML tag that create an embedded frame in the content of another page.
  ‣ This is the attack vector de jour for adversaries attempting to delivery content that exploits browser vulnerabilities.
  ‣ E.g., deliver crafted .jpg or malicious scripting

• The attack occurs when the adversary breaks into a webserver and places a IFRAME in legitimate content
  ‣ e.g., by sniffing passwords, recursively adding IFRAMEs

<iframe src=http://[REMOVED].info/counter style=display:none></iframe>
Drive by downloads

• Using a deceptive means to get someone to install something on their own (spyware/adware)

- Once you have one, then it starts downloading lots of others, their friends, …
- A personal favorite: *extortion-ware* -- pay us 40$ for our popup blocker, etc ….
- The real gambit is that they demand 40$ for the uninstall option
Web Systems Evolve ...

• The web has evolved from a document retrieval and rendering to sophisticated distributed application platform providing:
  ‣ dynamic content
  ‣ user-driven content
  ‣ interactive interfaces
  ‣ multi-site content content
  ‣ ....

• With new interfaces comes new vulnerabilities ...
Session Hijacking

• Virtual sessions are implemented in many ways
  ‣ session ID in cookies, URLs
  ‣ If I can guess, infer, or steal the session ID, game over
  ‣ Example, if your bank encodes the session ID in the url, then a malicious attacker can simply keep trying session IDs until gets a good one.
    http://www.mybank.com/loggedin?sessionid=11
  ‣ ... note that if the user was logged in, then the attacker has full control over that account.
  ‣ Countermeasure: randomized, confidential session IDs that are tied to individual host address (see cookies)
Preventing Web System Attacks

- Largely just applications
  - In as much as applications are secure
  - Command shells, interpreters, are dangerous
- Broad Approaches
  - Validate input (also called input sanitization)
  - Limit program functionality
    - Don’t leave open-ended functionality
  - Execute with limited privileges
  - Input tracking, e.g., taint tracking
  - Source code analysis
The new web-page

- Rendered elements from many sources containing *scripts*, *images*, and stylized by *cascading style sheets* (CSS)

- A browser may be compromised by any of these elements
Web-server APIs

- Web-servers often provide application extension APIs to which developers can build ...
  - ISAPI
  - Apache API
- Act as kinds of “kernel modules” for web-server
  - Web-server processes received inputs (URL, fields, etc.)
  - Passes result to custom code (typically, C code)
Application Frameworks

- Application frameworks are software stacks that implement a web application
  - Programmer adds domain-specific programming
  - Handle request handling and rendering
  - Quickly implement web apps without dealing the the nasty details of HTTP/HTML

- For example, the Zend framework implements a web application by processing incoming URLs
  - E.g., http://base/module/function
  - Zend accepts returned framework objects and renders them via internal API
  - Modify documents on the fly using AJAX scripts such as JavaScript
AJAX

- **AJAX**: asynchronous JavaScript and XML
  - A collection of approaches to implementing web applications
  - Changes the click-render-click web interface to allow webpages to be interactive, change, etc.
  - Examples: Google Gmail/Calendar, Facebook, ...
  - Hidden requests that replace document elements (DOM)
Attacks on web systems

- Web systems have replaced custom organization, enterprise and customer applications..
- ...this move is has led to many new attacks...
Cross-Site Scripting (XSS) - Persistent

• Assume the following is posted to a message board on your favorite website:

  Hello message board.
  
  <SCRIPT>malicious code</SCRIPT>
  This is the end of my message.

• Now a reasonable ASP (or some other dynamic content generator) uses the input to create a webpage (e.g., blogger nonsense).

• Now a malicious script is now running
  ‣ Applet, ActiveX control, JavaScript…
A non-persistent (or reflected) XSS attack targets pages immediately viewed by the user.

The bait is an innocent looking URL,
- Commonly in Email or on a neutral site.

The user clicks on a link and the content from a form or a URL is displayed on the resulting page.

If the resulting page doesn’t escape input, the script is executed.

Both persistent and non-persistent XSS are examples of injection attacks.
Injection

• Attacker that can inject arbitrary inputs into the system can control it in subtle ways

  ‣ *interpreter injection* - if you can get PHP to “*eval*” your input, then you can run arbitrary code on the browser ...  

  ‣ e.g., leak cookies to remote site (e.g., session hijacking)

    ```
    $INPUT = "Alice\;mail($to, $subject, $body)";
    ```

  ‣ *filename injection* - if you can control what a filename is in application, then you can manipulate the host

    • Poorly constructed applications build filename based on user input or input URLs, e.g., hidden POST fields

    • e.g., change temporary filename input to ~/.profile

    ```html
    <FORM METHOD=POST ACTION="../cgi-bin/mycgi.pl">
    <INPUT TYPE="hidden" VALUE="~/profile" NAME="LOGFILE">
    </FORM>
    ```
Browsers

• Browsers are the new operating systems

• Huge, complex systems that support
  ‣ Many document types, structures, e.g., HTML, XML, ...
  ‣ Complex rendering, e.g., CSS, CSS 2.0
  ‣ Many “program/scripting” languages, e.g., JavaScript
  ‣ Dynamic content, e.g., AJAX
  ‣ Native code execution, e.g., ActiveX

• Virtualized computers in a single program ...
Browser Security

• We don’t have the ability to control this much complexity, so we have to try other things ...
  ‣ Restricting functionality, e.g., NoScript
  ‣ Process Isolation, e.g., OP, Chrome

• Read: http://www.google.com/googlebooks/chrome/