



Defeating Web 2.0 Attacks without Recoding Applications

Amichai Shulman, CTO, Imperva Inc.

Goals and Agenda

- Detection and Mitigation of JS-Hijacking and CSRF Attacks
 - Attack intro
 - Code based solution
 - Gateway based solution
- Detecting Fraud Attempts that Exploit CSRF and JS-Hijacking Vulnerabilities

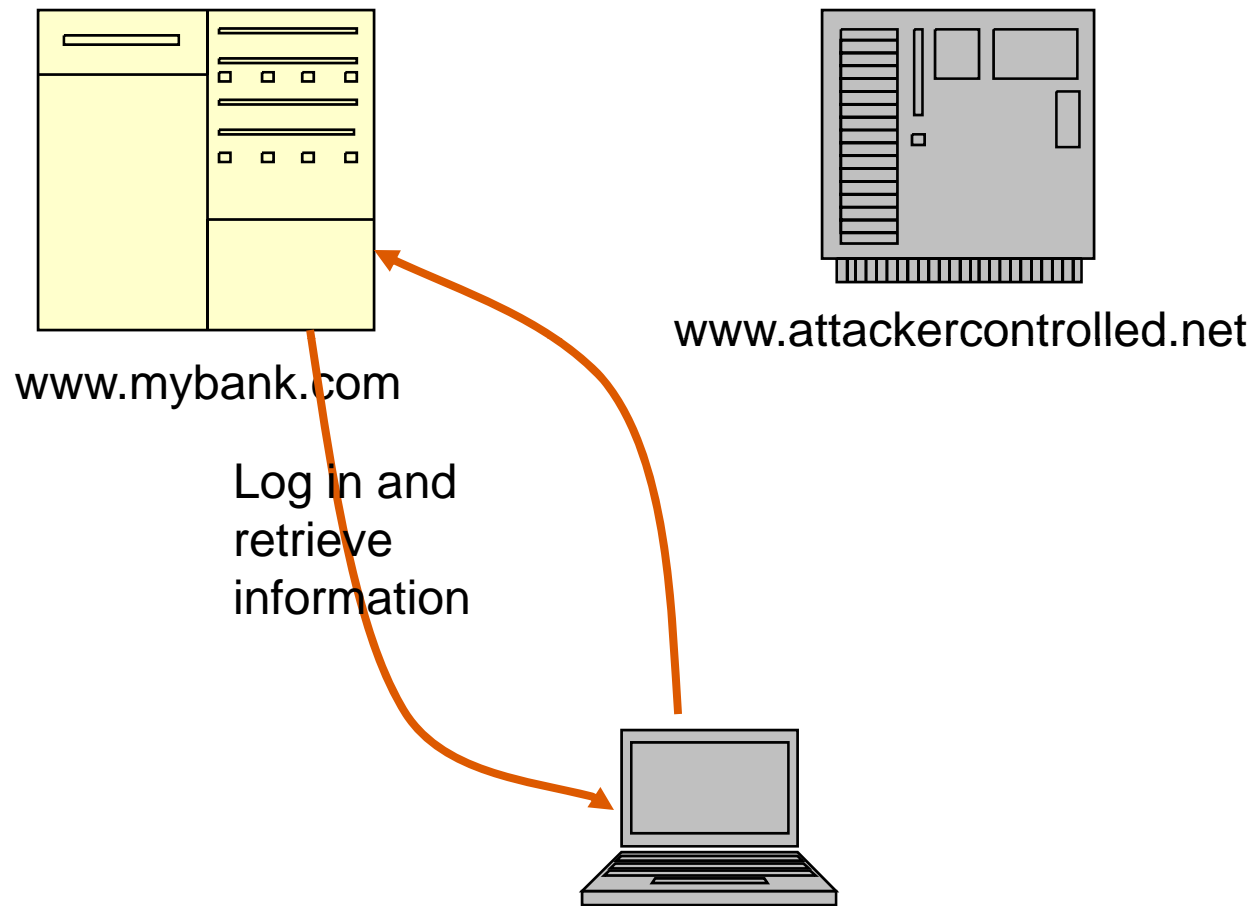
Why CSRF and JS-Hijacking

- JS-Hijacking is a newly discovered web 2.0 related vulnerability
- CSRF has been given a lot of attention lately. Experts predict that it's becoming the major issue in web security
- Traditional mitigation techniques are not suitable for cost-effective implementation
- To-date businesses are not properly protected against web frauds

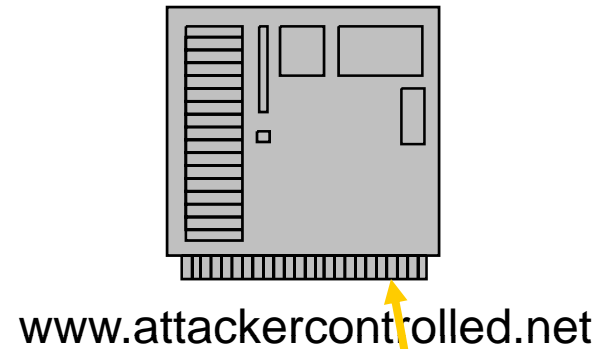
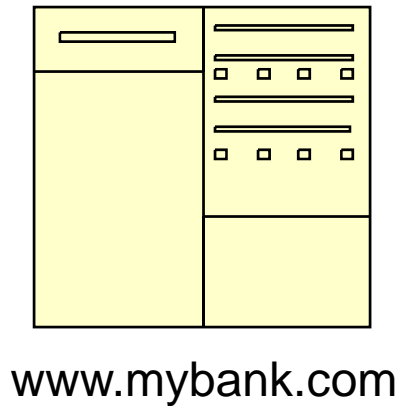
Introduction to JS Hijacking

- Introduced by Fortify on March 12, 2007
- Specific to applications who use Javascript as data transfer format – AJAX applications
- Abuses a loophole in the browser's Same Origin Policy
 - A script from any domain can be included and executed in the context of any other web site
 - If the script is used for application data transfer (it contains sensitive data in the form of JS arrays) that sensitive information can be accessed by code from a different domain
- Most notable example: gmail contact list

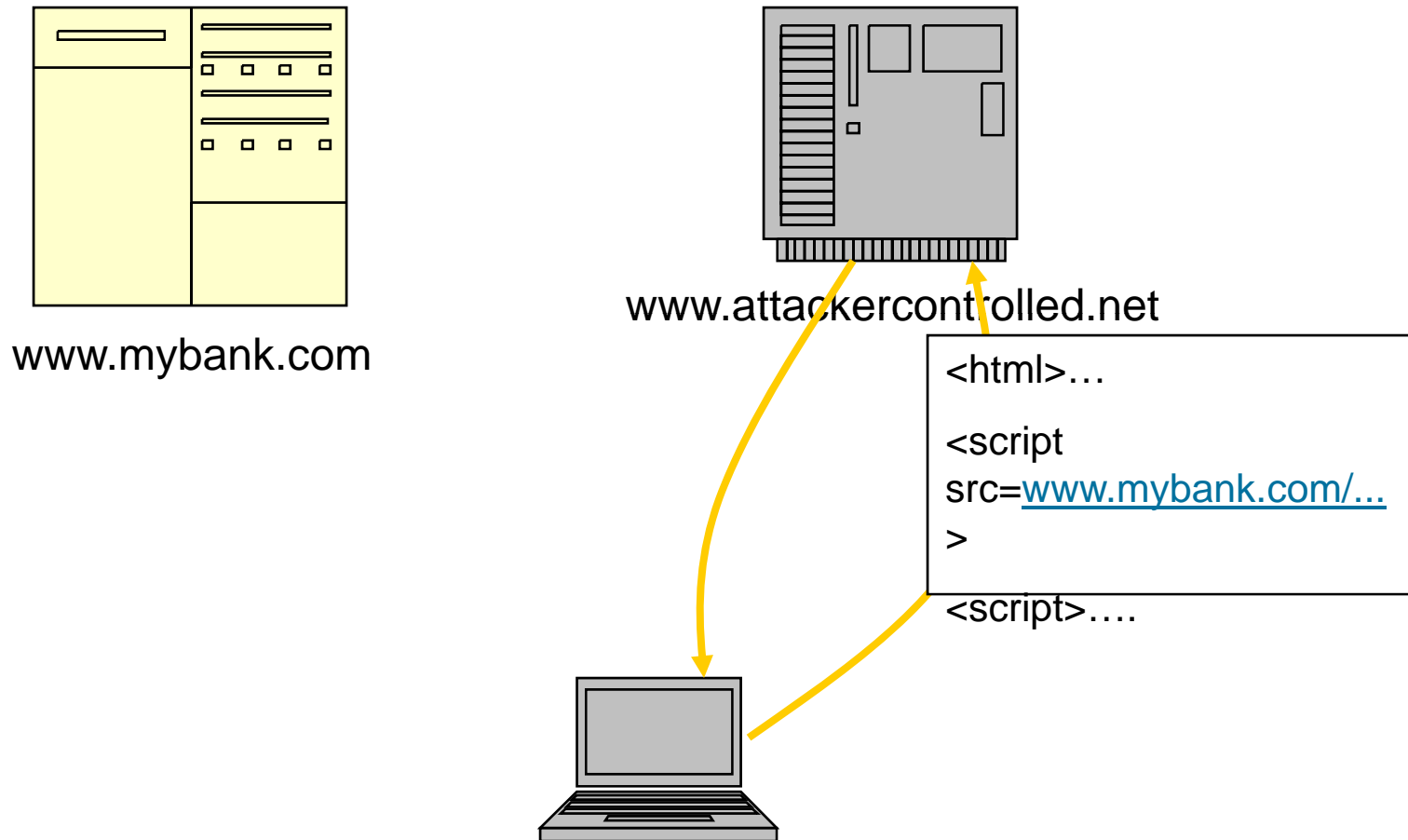
Introduction to JS Hijacking



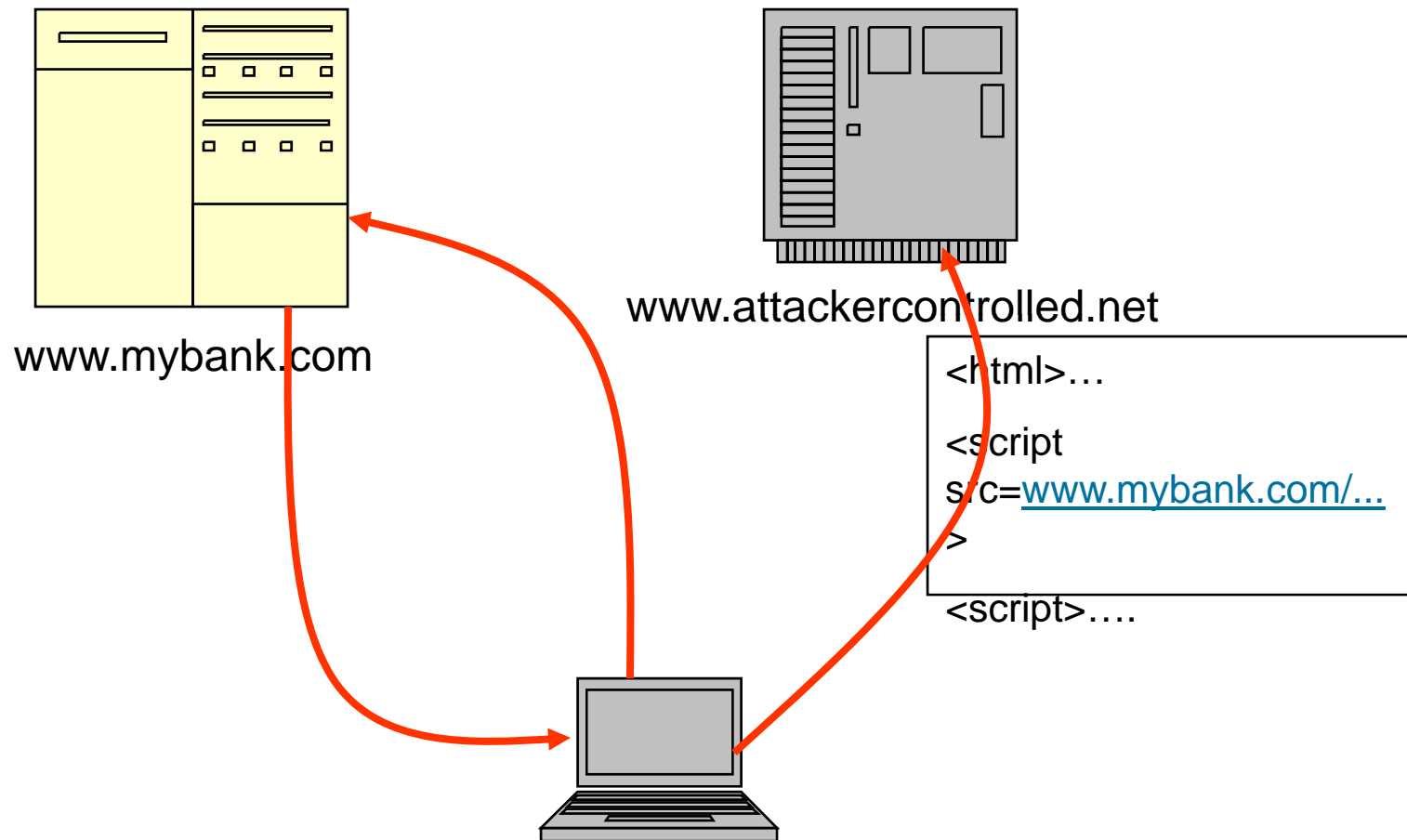
Introduction to JS Hijacking



Introduction to JS Hijacking



Introduction to JS Hijacking



Code Based Solutions for JS Hijacking

- Suggested by Fortify and others.
- Track a session nonce
 - Long random number generated per each new session (could be the session cookie)
 - Include the random number in each form / link
 - Validate the random element of each incoming form / link against the cookie / internal session pool
 - Reject requests with invalid random
- Change response format (use subverted Javascript)

Shortcomings of Code Based Solutions

- 3rd Party Components
 - Code for 3rd party components cannot be altered
- Deployed Applications
 - Modifying deployed applications is impractical extremely expensive and interrupts ongoing functionality
- Coding Practices
 - Enforcing secure coding practices is expensive and does not cover every loophole in an application

Shortcomings of Code Based Solutions (2)

- No Feedback for Victim
 - Victim is unaware of attack
 - Security personnel is unaware of attack until it is too late (the attacker has changed servers)
- No Support for Mashups
 - Access to application by Mashups would be rejected (unless more coding effort is put into the mashup)

Gateway Based Solution - Motivation

- Does not require code changes. Can be used for 3rd party components as well as deployed applications
- Does not disrupt application availability – no need to redeploy existing applications
- Protects all instances of JS Hijacking vulnerabilities within the application
- Continuous protection and detection regardless of new vulnerabilities being introduced into existing applications
- Provide detection as well as protection, including feedback for potential victims

Gateway Based Solution – Some Observations

- The “malicious” request is sent from the victim’s browser rather than an attacker’s client
- The “malicious” request is sent over an existing authenticated session (otherwise it would not make sense)
- The “malicious” request is invoked by a visit to an attacker’s controlled page outside of the application’s domain
- Allegedly “malicious” behavior can be the result of mashup usage
- Response is in Javascript format

Gateway Based Solution - Details

- Two step solution
 - Detect suspicious access
 - Interact with end user to resolve the event
- Detection
 - A request that belongs to an authenticated session
 - The domain in the “referrer” header field is not part of the application (either manual setup or dynamically profiled behavior).
 - The response is in Javascript format

Gateway Based Solution – Details (2)

- Interact with user
 - Prefix the Javascript code in the response with code to interact with user
 - Display an alert box that includes information about the source domain and requires user approval
 - If end-user approves then the response is further processed by the browser, otherwise processing of the response is stopped before sensitive information is available to the browser

Gateway Based Solution – Extensions

- The injected code can be further enhanced to accommodate a rich choice of user decisions:
 - Always block for the specific source domain
 - Always allow for the specific domain
 - Don't bother me anymore during this session
 - Never bother me again
- All the above can be implemented using browser controlled cookies inspected by the security gateway

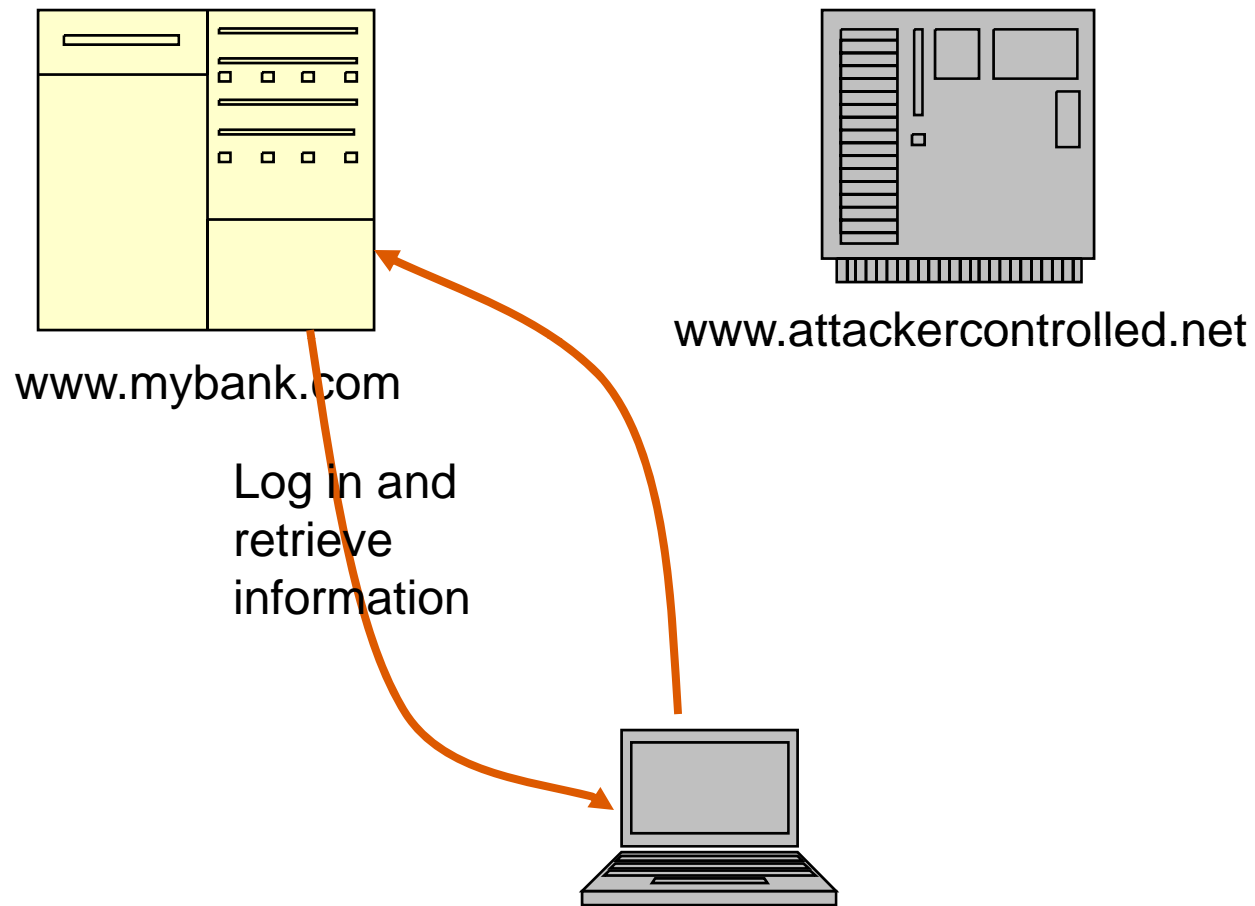
Gateway Based Solution – Fraud Detection

- The injected code can be further enhanced to report the decision back to the security gateway (either explicitly or through a cookie)
- The gateway behavior can be enhanced to detect attempted fraud and provide automatic reaction
 - A specific domain is reported by a variety of application users during a short period of time
 - The domain is flagged as malicious in the gateway's database
 - Further requests invoked by pages from the malicious domain are immediately discarded

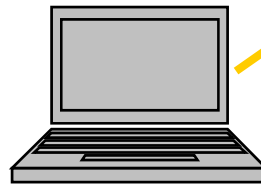
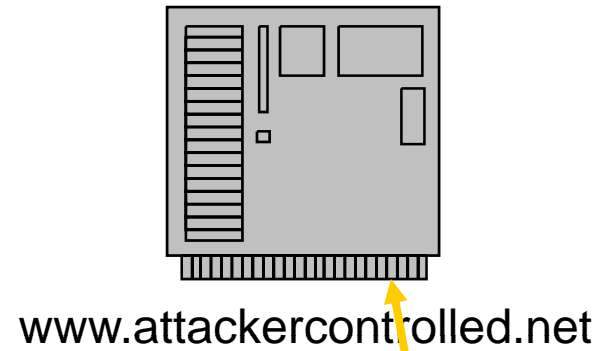
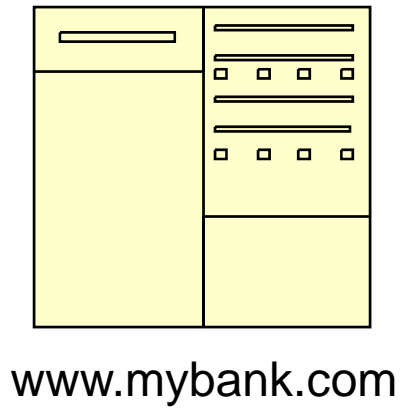
Short Introduction to CSRF

- Has been discussed extensively in previous sessions
- Abuses the server's trust in a client
- Take advantage of stateless modules with simple parameter interface
- Can be used to invoke application functionality
- Cannot be (generally) used to compromise sensitive information

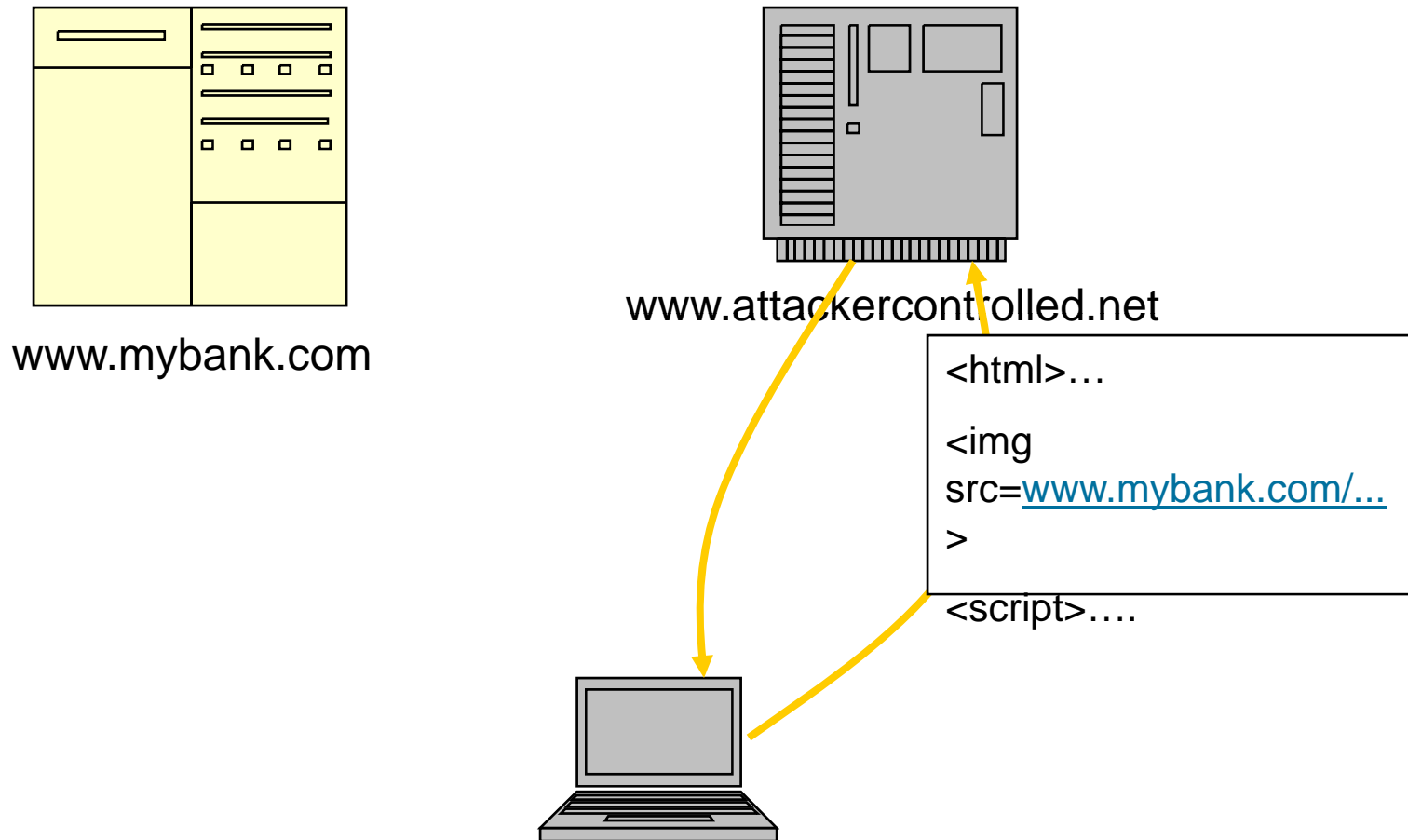
Introduction to CSRF



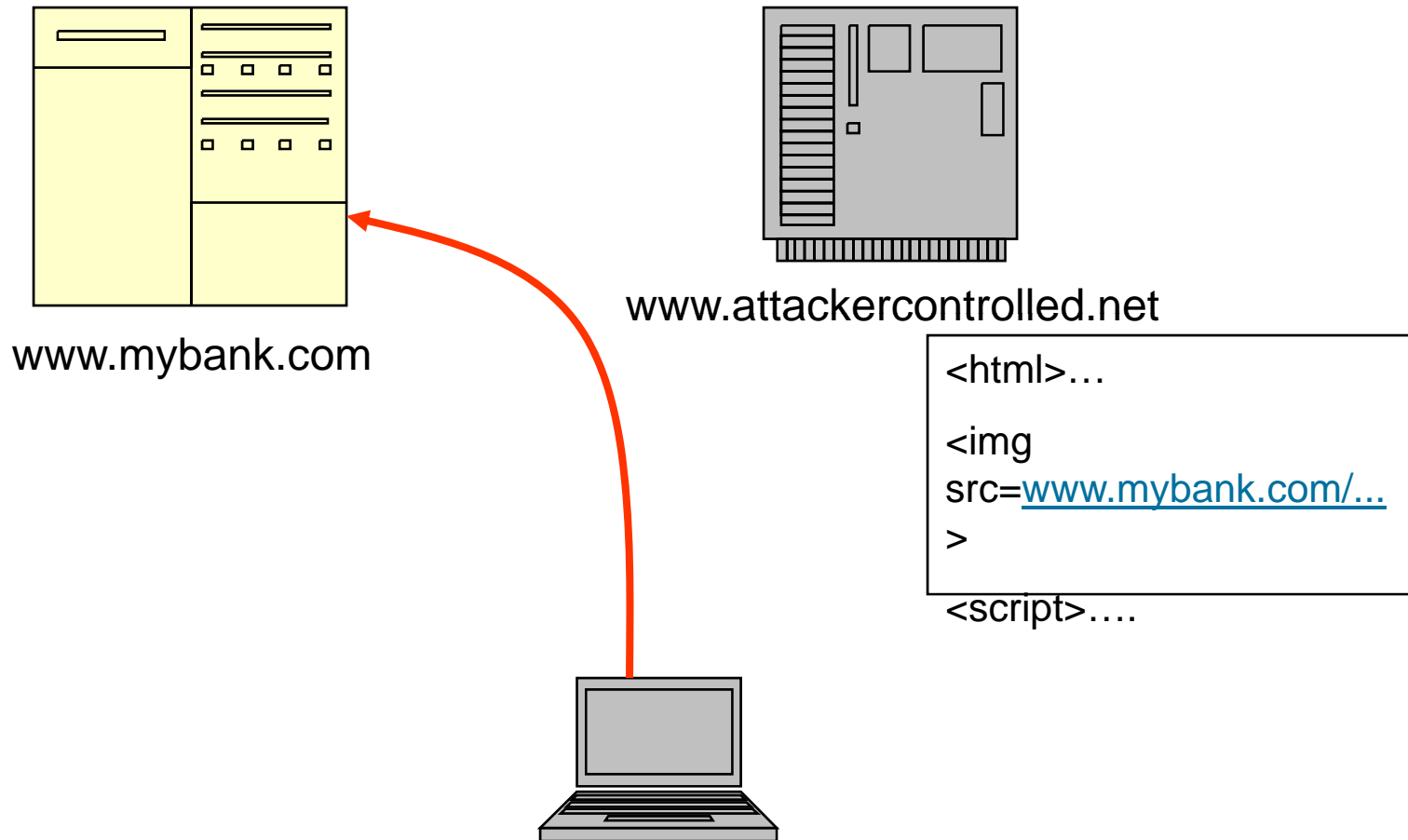
Introduction to CSRF



Introduction to CSRF



Introduction to CSRF



Code Based Solutions for CSRF

- Solution Types:
 - Track a session nonce (see earlier in this session)
 - Introduce CAPTCHA for sensitive modules
- Shortcomings
 - See earlier in this session regarding code based solutions
 - If CAPTCHA is introduced then redundant user interaction is required for each sensitive operation
 - Not always possible to introduce CAPTCHA without breaking the application

Gateway Based Solution - Challenges

- Detection of suspicious request is easy
 - A request that belongs to an authenticated session
 - The domain in the “referrer” header field is not part of the application (either manual setup or dynamically profiled behavior).
- However
 - Interaction with user must occur before request is conveyed to the server
 - Injecting code into response may break the application

Gateway Based Solution – Security Monitor

- Javascript code running on the client side on behalf of the security gateway
- Initialized when browser forms an authenticated session with the server through the security gateway
- Constantly waiting on commands from the security gateway
- Each instance of the security monitor is assigned a session nonce

CSRF and the Security Monitor

- When CSRF is suspected
 - Gateway internally flags the session as suspended
 - Sends HTTP 307 response to the original URL
 - Keep sending 307 response until a resolution is given through the Security Monitor
 - Alternatively store request locally and redirect using HTTP 302 to a void URL (together with a unique request id)

CSRF and the Security Monitor (2)

- Send code to Security Monitor
 - Display an alert message requesting user authorization (could be simple acknowledgement or CAPTCHA)
 - Code should include a digest of the request in question
 - User choice is communicated back to the gateway (together with the session nonce and the request digest)
- If user authorized the request the session is released from suspension and requests are conveyed to the server
- Otherwise, request is dropped (or HTTP 403 is sent back)

CSRF and Security Monitor - Extensions

- The injected code can be further enhanced to accommodate a rich choice of user decisions
- Fraud detection as described earlier

CSRF and Security Monitor - Optimization

- Problem
 - Implementing Security Monitor using standard protocol stack may incur a heavy burden on gateway (double the number of active connections)
- Observations
 - In an overwhelming portion of the sessions the Security Monitor is never used
 - In those session that actually use the Security Monitor it is used for an extremely small portion of the requests
 - Basically it's idle most of the time

CSRF and Security Monitor – Optimization (2)

- Solution
 - Implement a lightweight TCP stack on a dedicated gateway port
 - Each “lightweight” connection is represented by a four-tuple, sequence and ack numbers.
 - A single thread / process is used to monitor the port.
 - Requests are expected to be single frames and so are responses – thus buffering, reassembly and retransmissions are not required by server side stack

A Twist in the Plot – Nonce Injection

- Why not use a gateway to generate and track the nonce?
 - Requires that the nonce be introduced into each and every request
 - Requires understanding of application logic embedded into HTML responses
 - Seems highly inefficient
 - Bla bla bla

Nonce Injection

- The Little Engine Who Could
 - Most AJAX applications, whether they use a 3rd party framework or an in house developed infrastructure have their request generation code concentrated in an easily identifiable location
 - A dedicated script file
 - A distinctive piece of code within each response
 - It is easy to inject into the existing engine code additional code that embeds a session nonce into each request

Nonce Injection - Advantages

- A unified solution for an enterprise, regardless of the AJAX frameworks used
- Can be customized for in house developed infrastructure
- Can be applied to frameworks where the actual code cannot be changed
- Allows flexible configuration of protected paths vs. unprotected paths

Summary and Implications

- JS Hijacking and CSRF are two of the most common threats in Web 2.0 environments, expected to substantially affect the security of those applications
- Traditional, code based solutions are very costly and in many situations inappropriate for web applications

Summary and Implications

- Gateway based solutions can be designed to combine gateway side detection with end-user confirmation
- Gateway based solution are:
 - Cost effective
 - Allow fast and convenient integration into existing deployments and 3rd party components
 - Provide for continuous protection for the ever changing web applications
 - Can be expanded to detect large scale fraud



Thank You

Imperva, Inc.

3400 Bridge Parkway, Suite 101, Redwood Shores CA 94065

Sales: +1-866-926-4678 www.imperva.com

