CPSC156: The Internet Co-Evolution of Technology and Society

Lecture 22: April 17, 2007

Browser-based Security and Privacy Tools

Privacy and Security Problems

- Phishing
  - Spam directs users to spoofed websites
  - Malicious programs/websites steal info
- Passwords
  - Same password used at multiple websites
- Transaction Generators
  - "Hijack" user's session with a website

Stanford Anti-Phishing Projects

- http://crypto.stanford.edu/antiphishing
- SpoofGuard
  - Notify user about spoofed websites
- PwdHash
  - Transparently manage website-specific passwords
- SafeCache/SafeHistory
  - Prevent website from learning your prior behavior
- SpyBlock
  - Prevent unauthorized transactions

Spoofed Websites

- Why create them?
  - Steal private info (passwords, SSN, etc.)
- Users directed to fake websites
  - Easy to create website
  - Easy to imitate authentic websites
- Users typically enticed via spam
  - Easy to craft believable email
  - Easy to distribute email widely
- Examples: http://www.millersmiles.co.uk/

Traditional Indications

- Indications
  - Suspicious URLs
    - For example: http://www.ebay.com@129.170.213.101/
    - Requires user to read URL in address bar
  - Non-HTTPS URL
    - Most authentic websites requiring senstive information use HTTPS
    - Most spoofed websites don't use HTTPS
    - Requires user to read URL in address bar or notice the “lock” icon
- Problems
  - Users don't read carefully
  - Users don't understand what they see

SpoofGuard: Overview

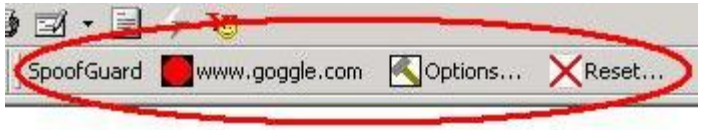
- Goal: Automate detection of spoofs
  - Don't rely on reactive measures (e.g., blacklists)
- Idea: Score each page visited
  - Score correlated with believe that webpage is a spoof
- Notify user of scoring results
  - Low suspicion: traffic light
  - High suspicion: force user to acknowledge popup
- Availability: Internet Explorer plugin

SpoofGuard: Scoring Criteria

- URLs and Links
  - Does the URL have a suspicious pattern?
- Images
  - Keep database of images and their domains
  - Are a page's images similar to ones from a different domain?
- Passwords
  - If page asks for a password, does it use HTTPS and have valid certificate?
- Referring Address
  - Was user referred from an email message (e.g., Hotmail)?
- Post Data
  - Store (hash of) posted data and domain
  - Is posted data same as data previously posted to a different domain?

SpoofGuard: Notification

- Traffic light in toolbar
  - Indicates score assigned to the page



- Popup notification
  - Forces user confirmation
  - Popup on any detected spoof; or
  - Popup only when user submits information
    - Intercepts form submission
    - Spoofs usually harmless when only viewing

The Same-Origin Principle

- Began with Netscape Navigator 2.0
  - "prevents document[s] or script[s] loaded from one origin from getting or setting properties of a document from a different origin."
    http://www.mozilla.org/projects/security/components/same-origin.html
- Why?
  - Information provided to/from a website should not be directly available to another website unless user explicitly provides it
- Applied to cookies (we've seen this before)

Types of Tracking

- Single-session / Multiple-session
  - Normal web features (e.g., via special URLs, cookies)
- Cooperative tracking
  - 3rd-party cookies, JavaScript, <META> tags
- Semi-cooperative tracking
  - Post link to external image on a forum
- Non-cooperative tracking
  - What can one learn without explicitly adding content to another site? We'll see...

SafeHistory and SafeCache

Content and DNS Caches

- Why store recently-used information?
  - Load pages faster, save bandwidth
- Timing attacks
  - Content cache
    1) User visits www.ebay.com
    2) User visits www.phishingsite.com, which measures how long it takes to load eBay logo
  - DNS cache
    1) User visits www.ebay.com
    2) User visits www.phishingsite.com, which measures how long it takes to lookup IP address for www.ebay.com

Loading From the Cache

- Assume http://www.mysite.com/index.html contains this HTML:

```
<img
src="http://i.microsoft.com/shared/core/1/images/ls.gif">
```

- Two different players
  - *Embedding site* (mysite.com)
    - The "carrier" for the image
  - *Hosting site* (microsoft.com)
    - Location in the network of the image being displayed

SafeCache: Overview

- Cached content is associated with embedding site
- Whats the difference?
  - Normally: Request for same hosted content is loaded from cache regardless of embedding site.
  - With SafeCache: Request for hosted content is loaded from ca*che only if* same embedding site previously cached it.
- Availability: Mozilla Firefox add-on

Visited Links

- Browser stores history of visited pages
- Visited links and unvisited links differentiated
  - Usually by color
  - Convenience to user
- But...
  - Font color can be read by page itself
    - JavaScript and Cascading Style Sheets
  - Phishing page can determine which websites the user has previously visited

SafeHistory: Overview

- Only two hosts can know if a page is visited
  - Host of the referrer
  - Host of the page itself
- Why only these two hosts?
  - Referrer could learn this information anyways (it can craft special hyperlinks)
  - The host of the page itself knows anyways (it can check its server logs)
- Availability: Mozilla Firefox add-on

Password Security

- Basic Problems
  - Many passwords easy to guess
    - Based on common words
    - Based on easily discoverable information (e.g., pet name, last name, etc.)
    - Traditional recommendation: use "random" combination of letters and numbers (hard to remember!)
  - Same password used at multiple websites
    - Stealing password from weakly-secured website gives access to account at highly-secured website
    - Traditional recommendation: use different password at each website (also hard to remember!)

Some Other Solutions

- Password list managers
  - Store usernames/passwords for each site
  - Cons: lack of portability, must consult list each time
- Limited-time Passwords
  - Example: RSA SecurID
    - Code on device changes every 60 seconds
    - User's password is combination of master password and code displayed on device
    - Cons: must carry device, typically only for single domain

PwdHash: Overview

- Let user remember a single "master" password
- Transparently convert password into site-specific password
- As a bonus, provides protection from common phishing attacks!
- Availability: Mozilla Firefox add-on

PwdHash: How It Works

1) Find all password fields on a page
   - `<INPUT type="password" ... >`
2) User enters '@@' before typing password
   - Signals browser to begin capturing password
3) Browser captures the user password and computes hash: $HMAC_{pwd}$(domain-name)
4) Hash is stored in password field and submitted to website in place of master password

PwdHash: Other Features

- Protection against common phishing attacks
  - Domain name is part of hash generation
  - Example:
    - $HMAC_{\text{"password"}}$(bankofamerica.com) = "y8JSLKDPFO"
    - $HMAC_{\text{"password"}}$(bankofamericas.com) = "pDVn5u7UYO"
- Usable when roaming
  - http://www.pwdhash.com/
  - Generates hash within the browser (via JavaScript)
  - Neither master password nor generated password are ever communicated over network

PwdHash: Why the '@@'?

- Consider the straightforward approach
  - Translate passwords when user leaves form field
  - Use domain name from target of the form
- But… webpages can execute code (JavaScript)
  - Monitor keyboard
  - Change form target before it is submitted
    - Before submission:

      `<FORM action="http://www.citibank.com/submit.cgi">`
    - After submission:

      `<FORM action="http://www.phishingsite.net/submit.cgi">`

PwdHash: Limitations

- Runs inside browser
  - No protection against DNS attacks
  - No protection against spyware
  - Limited protection for Flash

Is Password Security Enough?

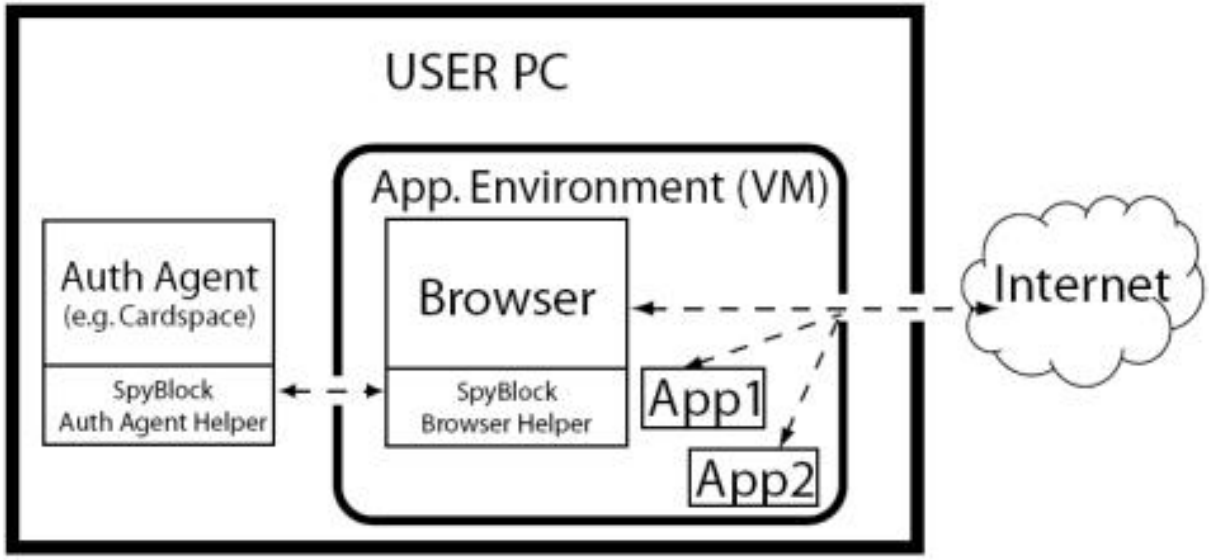
- Consider this scenario
  1) User logs into www.ebay.com
  2) Interacts with website as usual, possibly bidding on items and making purchases
- But...
  - Malicious software can send messages over authenticated session
  - These are called *transaction generators* (TGs)

How TGs Work

1) User logs into website with username and password
2) Website issues "session cookie" which is sent by the user with subsequent messages
3) TG can access this session cookie
4) TG initiates its own transactions using the session cookie

TG never needs to know the user's password!

SpyBlock: Overview



- Browser and all applications run within *virtual machine* (VM)
- User confirms transactions in trusted environment
- Availability: Mozilla Firefox add-on under Windows Vista

SpyBlock: The Pieces

- Virtual Machine
  - Essentially, an operating system running within another operating system
- Authentication Agent
  - Runs outside virtual machine, not alongside browser and other applications
  - Prompts user to confirm transactions
- Browser Helper
  - Allows browser to initiate transaction confirmation
  - *Cannot confirm transactions itself*

SpyBlock: Confirmation

1) Website requests confirmation (request accompanied with transaction details)
2) Browser helper passes transaction details to authentication helper
3) Authentication agent and website have shared key K (or they generate one if necessary)
4) Authentication agent computes hash:
   $T = HMAC_K(\text{transaction details})$
5) Authentication agent passes T to browser helper, which submits it to the website
6) Website can compute $HMAC_K(\text{transaction details})$ itself and verify against T

SpyBlock: Downsides

- Website must support SpyBlock transaction confirmations
- Though available for free, most people don't run virtual machines
- Security may be compromised as soon as user runs a *single* untrusted application outside virtual machine