

Keyjacking: Risks of the Current Client-side Infrastructure

John Marchesini, Sean Smith, Meiyuan Zhao
Department of Computer Science - Dartmouth College



I. Motivation

- Dartmouth PKI lab is building a PKI
- Do COTS PKI tools/appliances work?
- Some of our lab's prior work:
 - ★ Server-side SSL PKI [YeSm02]
 - ★ Digital signature tools like E-lock [KSA02]
- Today: client-side SSL PKI and browser-based keystores
- “Gawk at how much more frustrating IE proves to be”

The Fundamental Question

- *Does it work?*
- Can Claire (at the client) safely assume her private key was used only to authenticate services she intended, and was aware of?
- Can Victor (at the server) assume that, if a request was authenticated via client-side, the client was aware of and approved that request?
- Our environment: wide range of users, often shared computers, mostly Windows

II. Background

What Are We Protecting, Anyway?

1. Web Information Services - e.g. registering for classes, online ordering, digital library, etc.
 - Claire fills out an HTML form & Submits it
 - ★ GET method sends form data at end of URL
 - ★ POST method sends form data in a second request part
 - CGI script handles the form on the server
2. Other applications that use the browser keystore for client keys

Status Quo Authentication

- Client Address
- Passwords
- Cookies
- The clear message in the literature is that client-side is better
- In fact, maybe you should migrate your password-protected forms to client-side SSL

Browser-Based Keys

Netscape/Mozilla

- Public key is in *cert7.db*, private key in *key3.db*
- These are protected by a user-supplied password

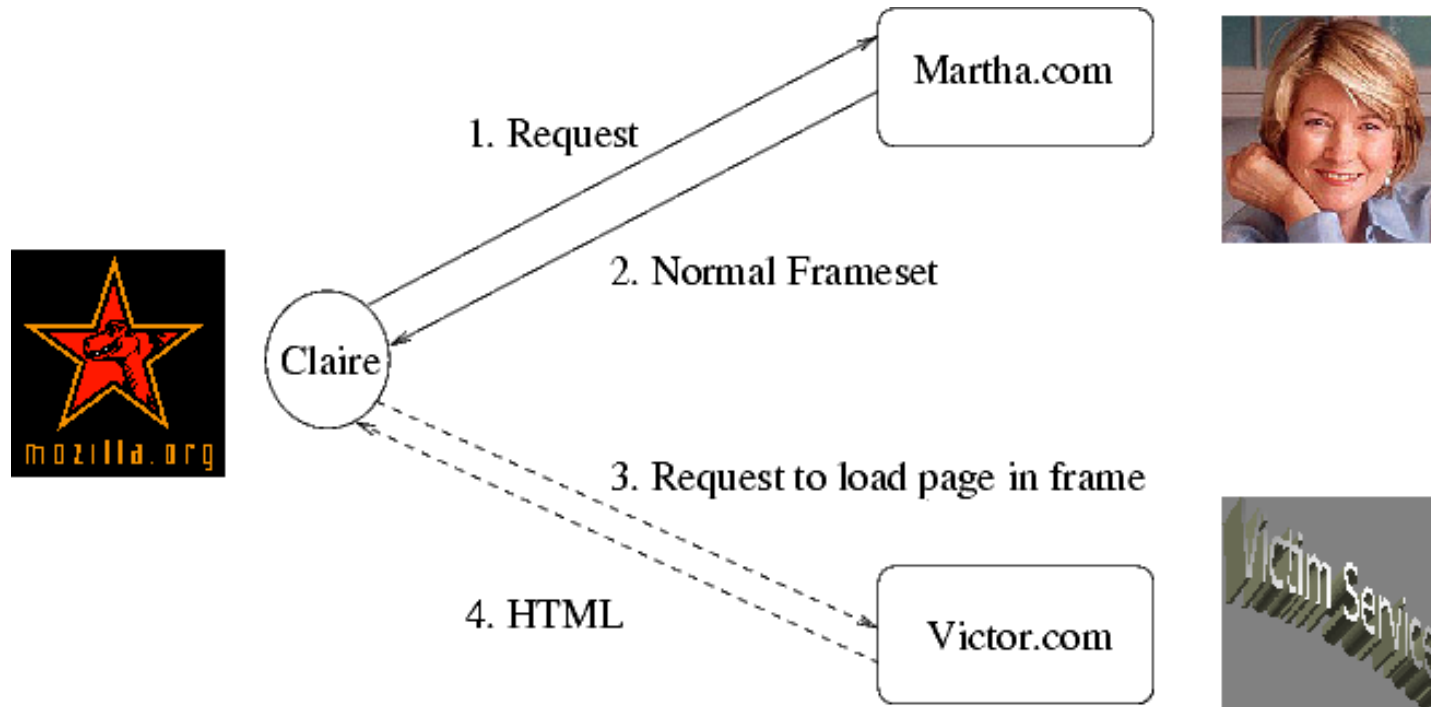
Internet Explorer

- IE stores keys in registry by default - *No password*
- Can export the private key to disk - password protected
- Since IE4, CryptoAPI generates/imports low/medium/high security keys

Historical Vulnerabilities

- Netscape relatively unchanged - historical = current vulnerabilities
- Registry stored keys - “Offline NT Password & Registry Editor”
- Peter Gutmann’s *breakms* - dictionary attack on exported keys
- We didn’t see attacks on medium or high security keys

III. Our Experiments: Using Keys

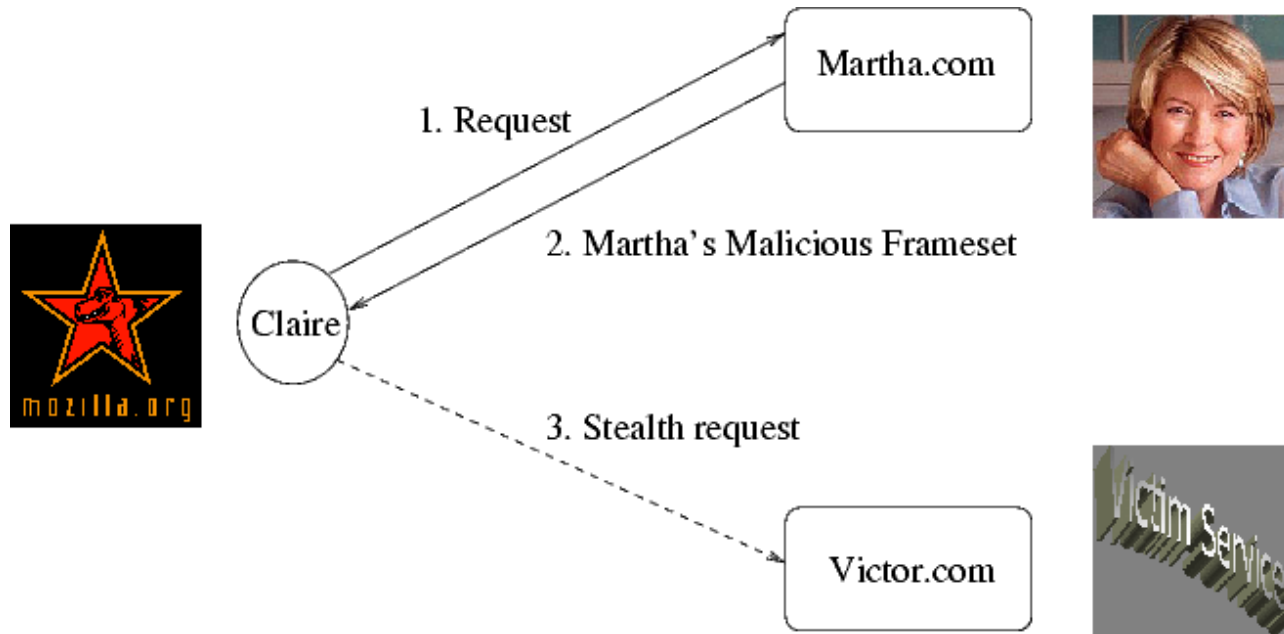


- `framesets` enable servers load different URLs into frames
- It is possible to hide frames
- Martha can tell the browser to load an SSL page from Victor's server and *browser often only reports Martha's cert*

Using Keys Continued

- If Victor requires client-side authentication:
 1. Will Claire's browser authenticate to Victor?
 2. Will the browser ask Claire, or even tell her?
 3. Will Claire notice?
- In many out-of-the-box configurations we tried, the browser happily uses Claire's private key *without telling her*
- Semantics sometimes change depending on how many keys Claire has, and whether she's armed them
- Request could be a form submission via GET or POST (requires Javascript)

Implications



- In many configurations, SSL-auth'ed requests can be forged
- With Netscape/IE, it's easier to forge a request if Claire goes to Martha.com after having done some service at Victor.com
- If Martha claims she also requires client-side auth, users may even ignore browser signals

IV. Stealing Keys from Foolish Users

- Poor UI to trick users into giving up their keys and password
- Dartmouth Authentic Really Secure Service (DARSS) - a website which advertises a service which requires a private key
- If successful, use the password to open the encrypted file containing the private key using NDBS or OpenSSL

Stealing Netscape Keys from Wiser Users

- Server page which pops the Netscape password prompt
- Need to get the *cert7.db* and *key3.db* files from the user's home directory
- This requires us to compromise the user's account or have an executable which sends us the files

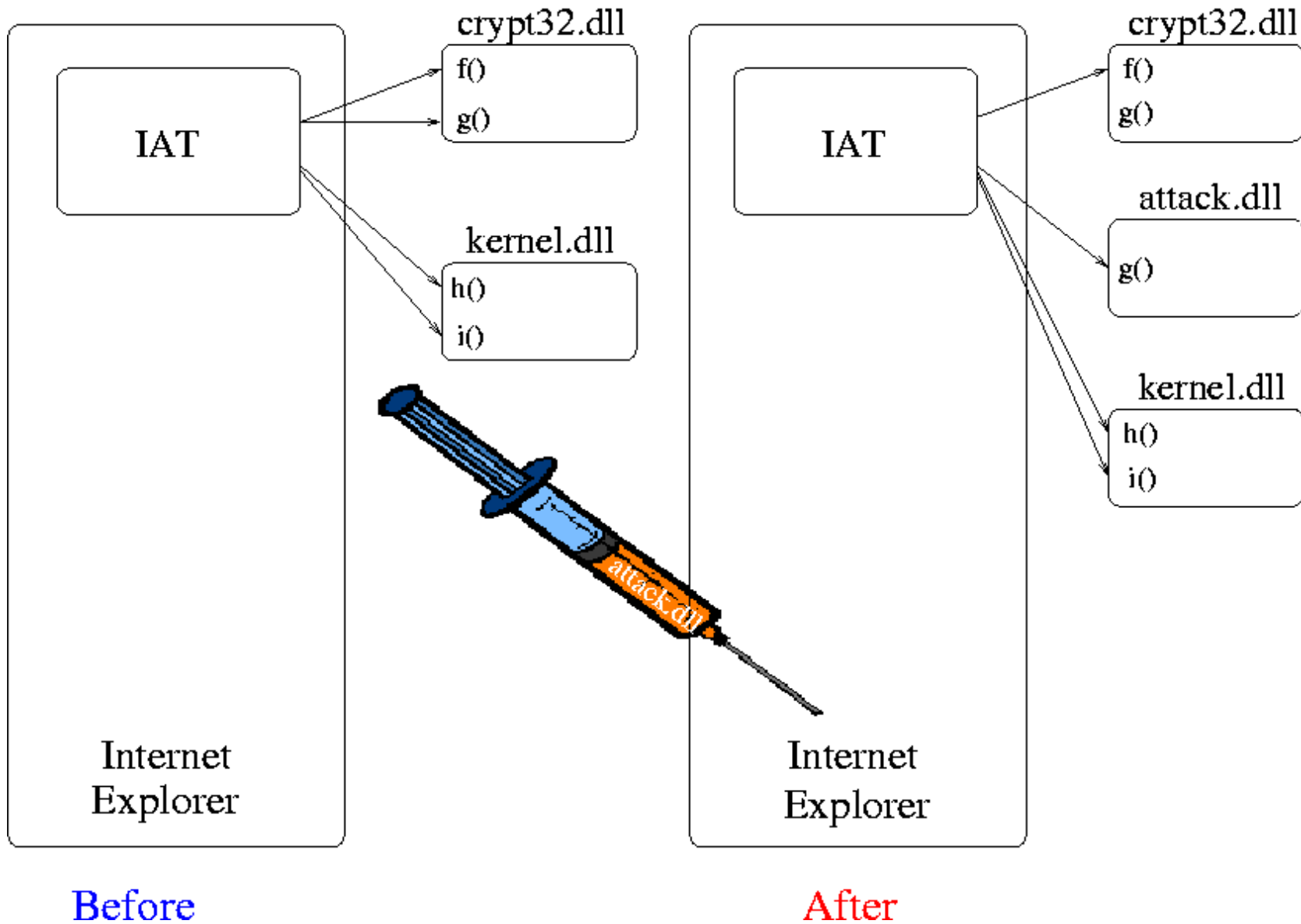
Stealing Microsoft Low Security Keys

- In 1998 Peter Gutmann was concerned about `CryptExportKey()`
- With a low security key, the key comes right out of the registry to any process with the user's privilege
- This trick still works
- Need a small executable on the user's machine

Stealing Microsoft Medium/High Security Keys

- Couldn't find any attacks against medium/high security keys, so we adapted some techniques from the gaming world
- *API Hijacking* intercepts calls from a process (e.g. IE) to an API (e.g. CryptoAPI)
- Man-in-the middle attack on the code level - any calls to the CryptoAPI functions we choose pass through our code first
- *DLL Injection* Inject our attack dll into IE's address space (via an executable which registers a Windows Hook)
- The attack dll overwrites IE's Import Address Table (IAT) to point to its functions instead of CryptoAPI's

Stealing Microsoft Medium/High Security Keys



Stealing Microsoft Medium/High Security Keys

- The end result: when IE accesses the keys (e.g. via `CryptSignMessage()`), we intercept the calls
- *We never trigger the extra alert* - the user is expecting to see some UI when `CryptSignMessage()` is called
- We could even be polite and complete the request

V. Short-Term Countermeasures

- Careful programming - “right thing to do” is hard to find
- Hidden form fields - after authentication, construct a form with requester’s name, etc., sign it, and check for this field on submission
- Use SSL `Keep-Alive` to re-negotiate periodically
- Use medium/high security keys in IE, and make non-exportable
- Moving keys to a token separates key from some OS weaknesses
- Educate users about browser configurations and implications

Long-Term Countermeasures

- Trusted paths from user to browser [YeSm02]
 - ★ Secure attention key - e.g. an `authenticate` tag
 - ★ Reverse Turing Tests
 - ★ Informing users what part of the screen is asking for what
- Browsers are too unclear to be the cornerstone of a secure system - remember Kiko's pile of dialog boxes?
- Tokens with UI - private keys are too important to be left on general purpose desktops - so is their interface

VI. Summary

- Back to our question: *Do the PKI tools/appliances work?*
- We continually find they don't
- We found that client-side authentication does not necessarily authenticate the person - http endpoints
- We found that browser keystores are too weak to be the foundation of secure systems
- There's a mismatch between "beautiful math" and permeable computers

The Big Picture

- What is the mental model that people have of when browsers will use private keys, and for what purpose?
 - ★ naive users, smart users
 - ★ system programmers
 - ★ policy makers
 - ★ for various browsers
 - ★ and various options
 - ★ and various surfing patterns
- Do these models match what the browser really does?
- If not, client-side PKI can't enable reasonable trust judgments
- Perhaps we have a serious systemic problem here

Thanks

- Thanks to Internet2, Mellon Foundation, The US Dept. of Justice, and the NSF for their support