NoScript, CSP and ABE: When The Browser Is Not Your Enemy

Giorgio Maone

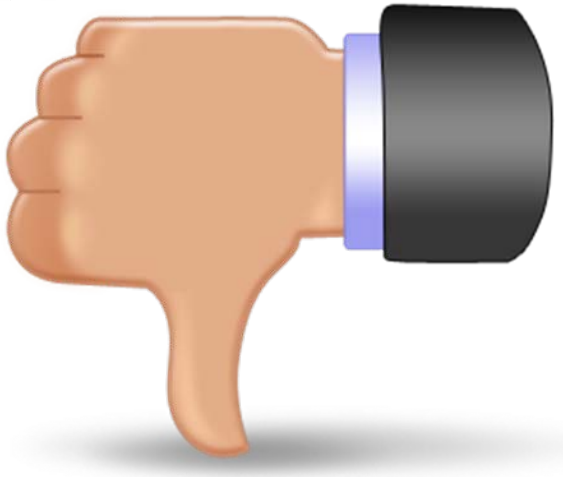
CTO, NoScript lead developer
InformAction

OWASP-Italy Day IV
Milan
6th, November 2009

Who Am I?

- ✓ Software developer
- ✓ Hacker
- ✓ Firefox contributor
- ✓ Member of the Mozilla Security Group
- ✓ Author of the NoScript browser add-on
- ✓NoScript user ;)

Can Browser Tech Mitigate WebApp Vulns?

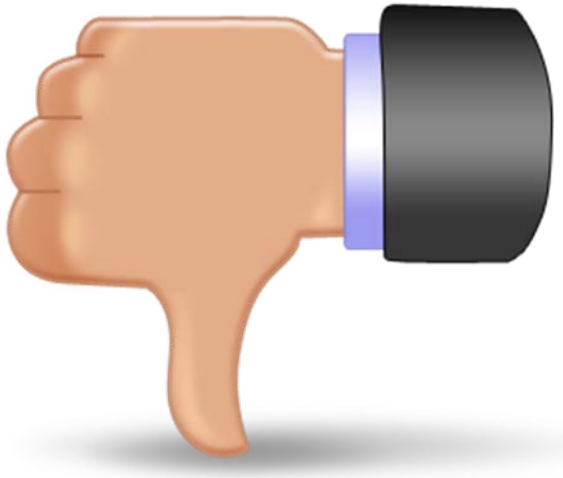


Before NoScript

No, it cannot: they're server-side issues.

We can only wait for web devs to fix them.

Can Browser Tech Mitigate WebApp Vulns?



Before NoScript

No, it cannot: they're server-side issues.

We can only wait for web devs to fix them.

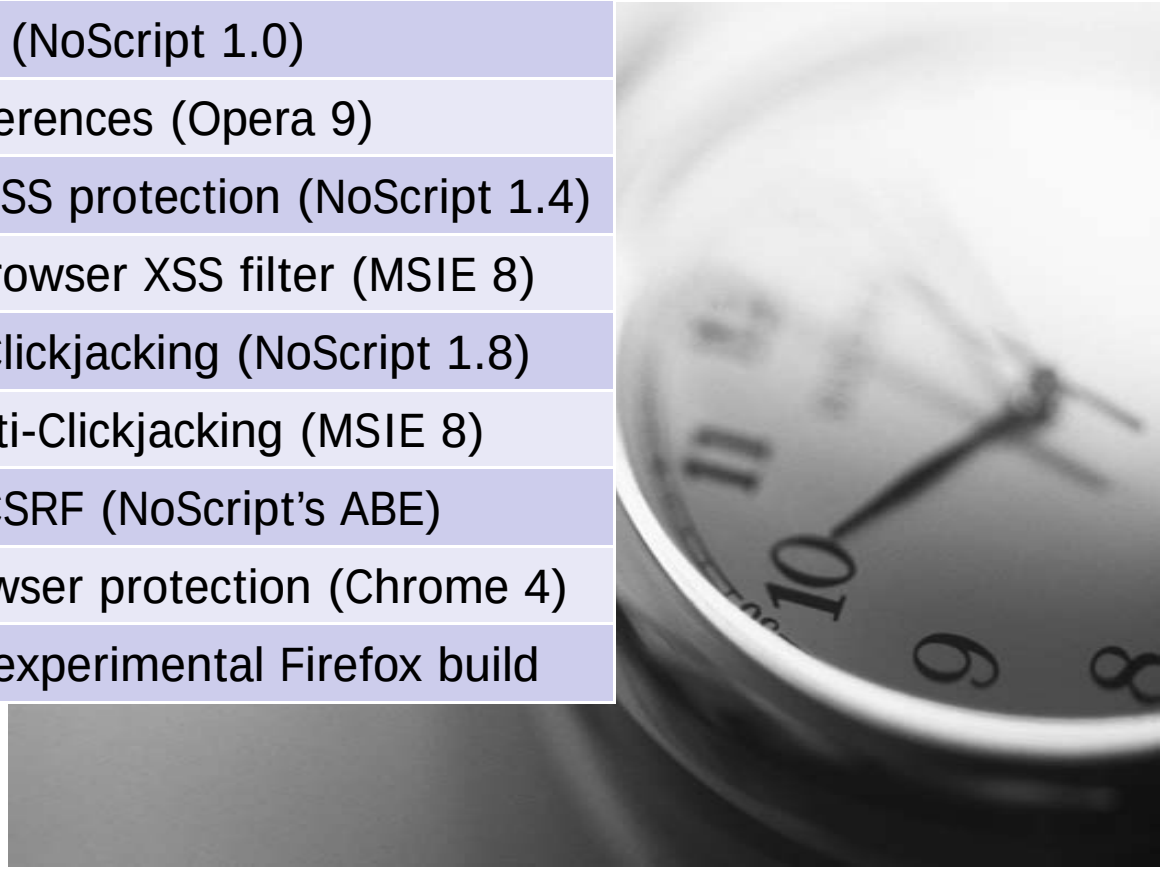
After NoScript

Well, it might... It can... **It should**!

Web developers still need to fix bugs and develop safely, but browser technology can and should help users to stay safer.

Timeline of Proactive Browser Security

Dec 1995	Same origin policy (Netscape 2)
Jun 1997	Security Zones (MSIE 4)
May 2005	Easy whitelisting (NoScript 1.0)
Feb 2007	Site specific preferences (Opera 9)
Mar 2007	Client-side anti-XSS protection (NoScript 1.4)
Jul 2008	Mainstream in-browser XSS filter (MSIE 8)
Oct 2008	Client-side anti-Clickjacking (NoScript 1.8)
Jan 2009	Server-driven anti-Clickjacking (MSIE 8)
Jun 2009	Client-side anti-CSRF (NoScript's ABE)
Sep 2009	More XSS in-browser protection (Chrome 4)
Oct 2009	1st CSP-enabled experimental Firefox build



Beyond the Same Origin Policy

- Guarding Cookies and JavaScript since 1995
- The only universal active content restriction for one decade
- Circumvented through:
  - ✓ Browser bugs
  - ✓ Plugin bugs
  - ✓ Web application flaws (XSS, CSRF, Clickjacking…)
- Easily compromised by careless mashups

Disabling Active Content

Pros:

- ✓ Default mitigation for most unpatched vulnerabilities in browsers & plugins
- ✓ Narrows the attack surface
- ✓ Prevents casual browsing surprises

Disabling Active Content

Pros:

- ✓ Default mitigation for most unpatched vulnerabilities in browsers & plugins
- ✓ Narrows the attack surface
- ✓ Prevents casual browsing surprises

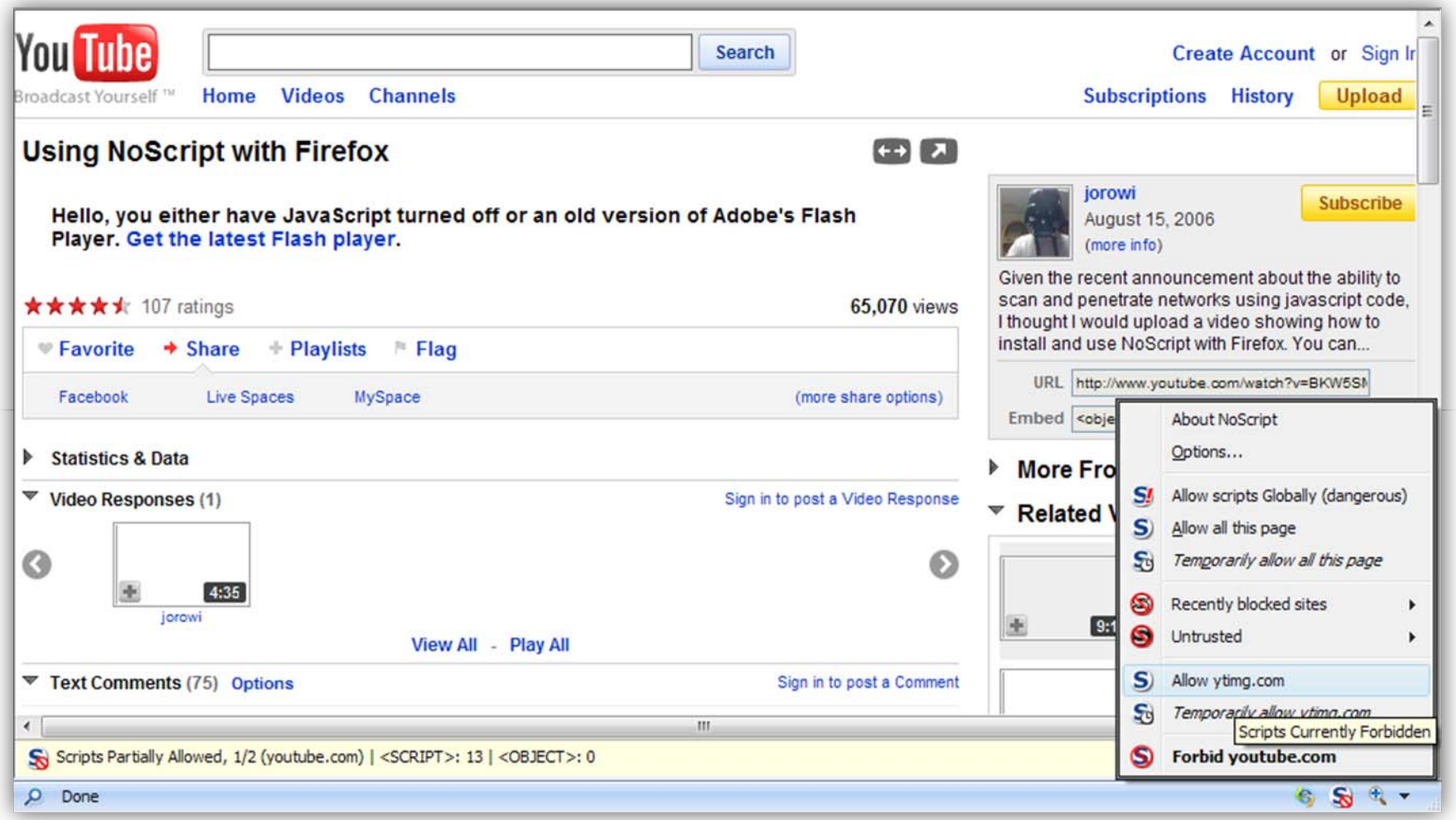
Cons:

- ✓ Many modern websites don't work properly
- ✓ Some web application security features are disabled as well (e.g. frame busting)
- ✓ Users will work against this policy

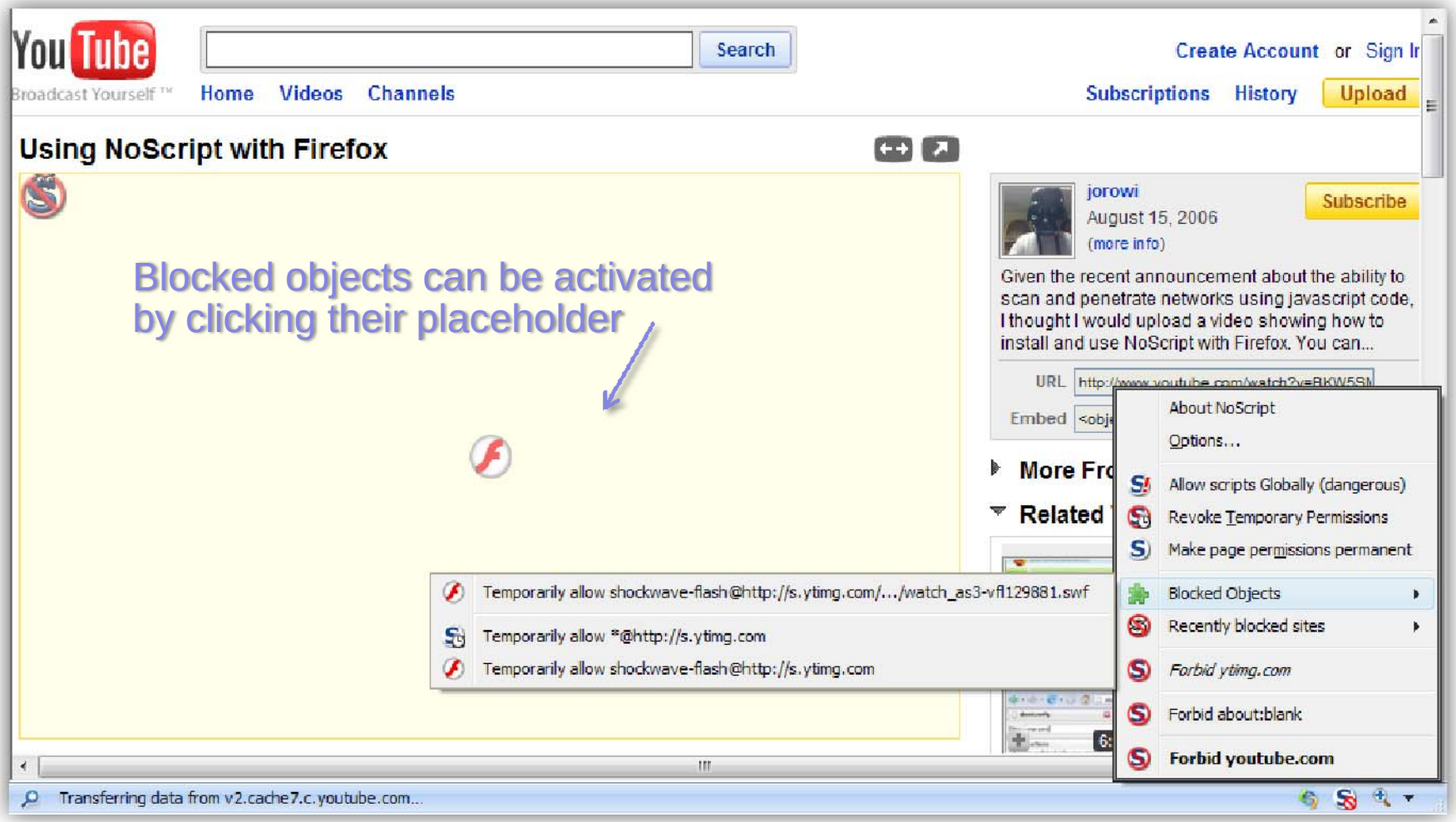
Enters NoScript...

- ✓ Default deny, easy allow
- ✓ Never blocks users with modal prompts
- ✓ Does not encourage “allowing everything”
- ✓ Makes mixed origins explicit
- ✓ Emulated JavaScript navigation
- ✓ Scriptless frame-busting
- ✓ Surrogate scripts
- ✓ One-click activation for embeddings

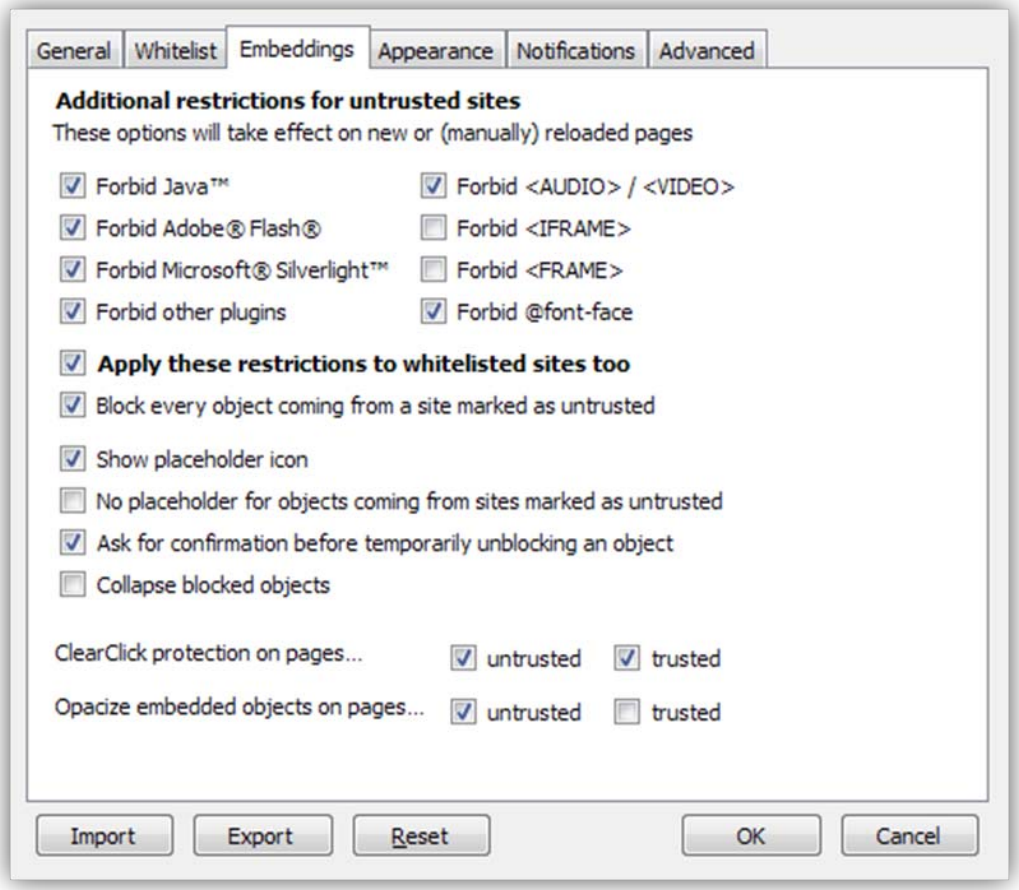
NoScript's Main UI



NoScript and Embedded Content (UI)

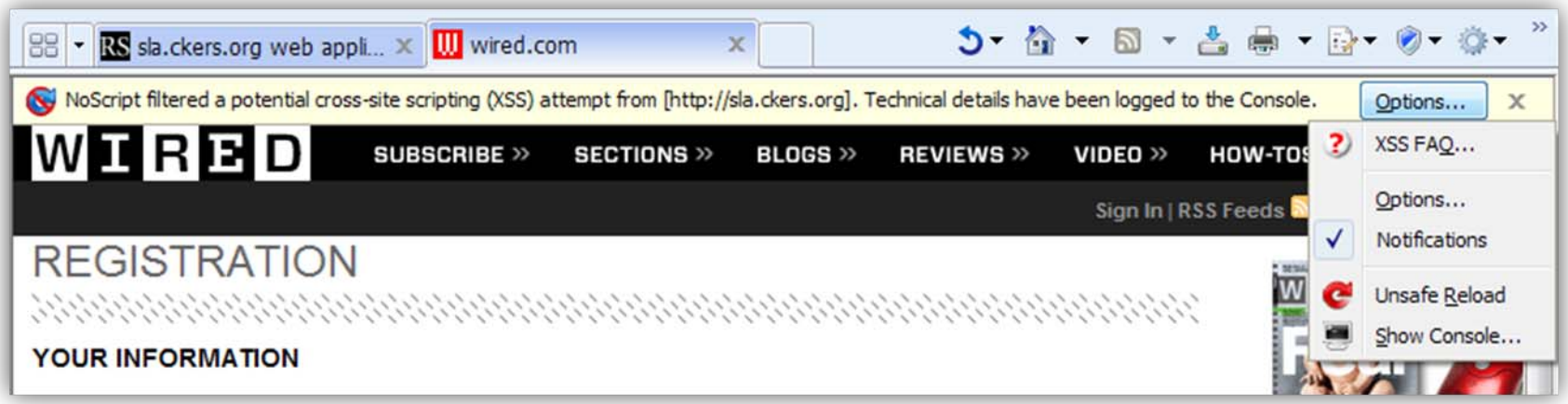


NoScript and Embedded Content (Options)



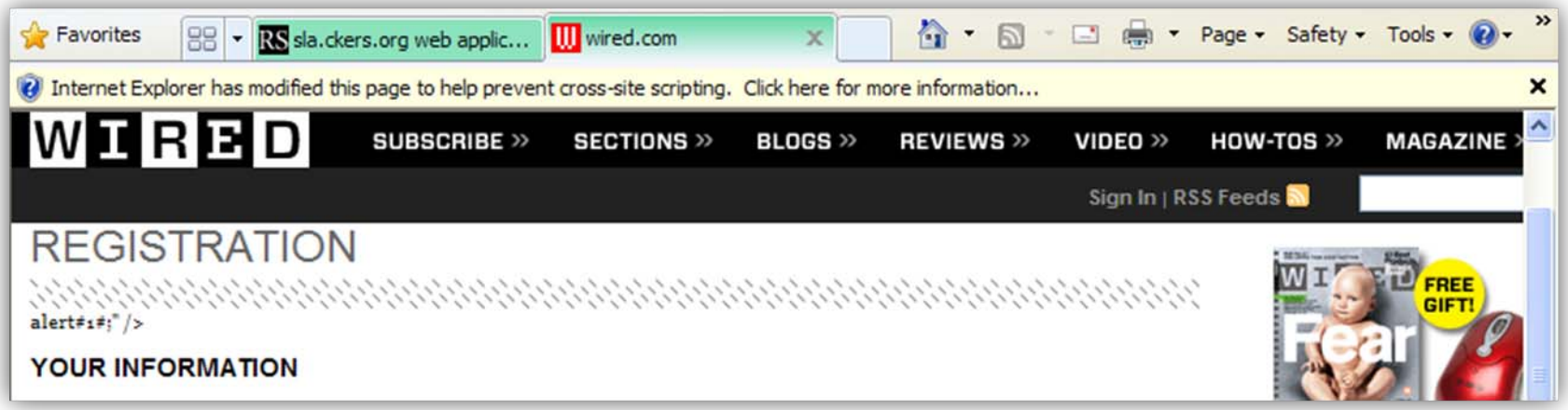
NoScript's Anti-XSS Protection

- ✓ Filters suspect requests (both JavaScript and HTML injections)
- ✓ UI is not modal and neutralization is not blocking
- ✓ Allows users to override ("Unsafe Reload")
- ✓ Works against all kinds of non-persistent XSS attacks (including DOM-based ones) and HTML injections



MSIE 8's XSS Filter

- ✓ Patches suspect responses (may add vulnerabilities)
- ✓ UI is not modal and neutralization is not blocking (impressively resembling NoScript, albeit with less rich UI)
- ✓ Allows server to override
- ✓ Does not work against DOM-based attacks and scriptless HTML injections



Chrome 4's XSS Filter

- ✓ Checks scripts in response against request fragments
- ✓ No UI
- ✓ Disables JavaScript on suspect pages (blocking neutralization, may be exploited to force a scriptless page)
- ✓ Allows server to override (same mechanism as MSIE 8)
- ✓ Does not work against DOM-based attacks and scriptless HTML injections
- ✓ Many bypasses found so far

Client-side XSS Protection Matrix

	NoScript	MSIE	Chrome
Type 0	Yes	No	No
Type 1	Yes	Yes	Yes
Type 2	No	No	No
HTML	Yes	No	No
UI	Yes	Yes	No
Non-blocking	Yes	Yes	No
Server override	No	Yes	Yes
User override	Yes	No	No

Content Security Policy (CSP)

Overview

- Declarative server-driven anti-XSS framework
- May require massive website changes
- Policies are pushed through HTTP headers
- Idea by RSnake, Design & Implementation by Mozilla (G. Markham, B. Sterne, S. Stamm)
- Just an experimental build so far, but interest from Microsoft and other parties
- Partially overlapping with other proposals (e.g. `X-Frame-Options` and `Strict-Transport-Security`)

Content Security Policy (CSP)

Features

- Effective against Persistent and Non-Persistent XSS (except DOM-based)
- Allows site admins and developers to:
  - ✓ Restrict JavaScript execution on the page, either totally or by disabling inline scripts and runtime evals
  - ✓ Specify script, stylesheet, media and embedding sources which are allowed for inclusion (whitelist)
  - ✓ Restrict frame hierarchies (like `X-Frame-Options`)
  - ✓ Force HTTPS (like `Strict-Security-Transport`)
- Reports violations to a configurable URL

Content Security Policy (CSP)

Deployment

- Each HTTP response must include one or more `X-Content-Security-Policy` headers
- The following restrictions are enforced (possibly requiring scattered page code changes):
  - ✓ No inline script block or event handler attribute
  - ✓ No runtime string evaluation (`eval()`, `new Function()`, `setTimeout()/setInterval()`...)
  - ✓ No `javascript:` / `data:` URIs
- Scripts from whitelisted sources must be served with content-type `application/(javascript|json)`

Content Security Policy (CSP)

Directives

- `allow` *(defines default for all types)*
- `options (inline-scripts/eval-scripts)`
- `img-src`
- `media-src`
- `script-src`
- `object-src`
- `frame-src`
- `font-src`
- `xhr-src`
- `frame-ancestors`
- `style-src`
- `report-uri`
- `policy-uri`

- ✓ Multiple headers/directives intersection
- ✓ `X-Content-Security-Policy-Report-Only`
- ✓ HTTPS enforcement (superseded by STS)

Content Security Policy (CSP)

Policy samples

- **Example 1:** Site wants all content to come from its own domain:

```
X-Content-Security-Policy: allow 'self'
```

- **Example 2:** Auction site wants to allow images from anywhere, plugin content from a list of trusted media providers, and scripts only from its server:

```
X-Content-Security-Policy: allow 'self'; img-src *; object-src media1.com
                media2.com *.cdn.com; script-src trustedscripts.example.com
```

- **Example 3:** Admins want to deny all 3rd-party scripts for the site, and a project group also wants to disallow media from other sites:

```
X-Content-Security-Policy: allow *; script-src 'self‘
X-Content-Security-Policy: allow *; script-src 'self'; media-src 'self';
```

- **Example 4:** Online payments site wants to force all of the content in its pages to be loaded over SSL (should be much better using STS):

```
X-Content-Security-Policy: allow https://*:443
```

Strict Transport Security (STS)

- W3C draft proposed by Paypal
- Implemented by NoScript, soon by Chrome, interest from other browser vendors (Microsoft, Mozilla)
- Both Paypal and Ali Baba are deploying it
- Very simple yet effective

```
Strict-Transport-Security: max-age=31536000; includeSubDomains
```

NoScript can also force HTTPS on user-choosen sites

Application Boundary Enforcer (ABE)

Overview

- Declarative anti-CSRF mechanism
- Both user-driven and server-driven
- Rules priority: SYSTEM, USER, Subscriptions, Site
- **https**://domain.com/**rules.abe**
- Simple firewall-like rules definition syntax
- Open source specification and reference implementation
- Currently available as a NoScript component, it can be implemented as a proxy or a server-side component as soon as CORS is finalized and adopted by browsers

Application Boundary Enforcer (ABE)

Rules definition syntax

```
Site <resource> [<resource> ...]
<action> [<method>...] [from <resource> [<resource>...]]
[<action> [<method>...] [from <resource> [<resource>...]]
...]
```

- **Resource**: either an URL pattern (glob, regexp), `LOCAL`, `SELF`, or `ALL` (*)
- **Action**: either `Accept`, `Deny`, Anon(ymize), or `Sandbox`
- **Method**: either a “real” HTTP method, or SUB, or ALL (default)

Application Boundary Enforcer (ABE)

Ruleset example

```
# This one guards the LAN, like LocalRodeo (a SYSTEM rule in NoScript)
Site LOCAL
Accept from LOCAL
Deny
# This rule defines normal application behavior, allowing hyperlinking
# but not cross-site framing and POST requests altering app status
Site *.somesite.com
Accept POST SUB from SELF https://secure.somesite.com
Accept GET
Deny
# This one guards logout, which is foolish enough to accept GET and
# therefore we need to guard against trivial CSRF (e.g. <img>)
Site www.somesite.com/logout
Accept GET POST from SELF
Deny
# This one strips off any authentication data (Auth and Cookie headers)
# from requests outside the application domains, like RequestRodeo
Site *.webapp.net
Accept ALL from *.webapp.net
Anonymize
```


Application Boundary Enforcer (ABE)

Deployment

- SYSTEM ruleset, built-in (currently containing just the *LocalRodeo* replacement rule)
- USER ruleset, customizable by users
- Subscription rulesets, updated daily from remote trusted sources
- Site rulesets (rules.abe), loaded before first HTTPS request and cached for one day (at least) or more (if cache-affecting headers say so)
- Why a file (log spam) rather than headers? CSRF needs to be blocked *before* the request reaches the server: we just can't wait for a response header…

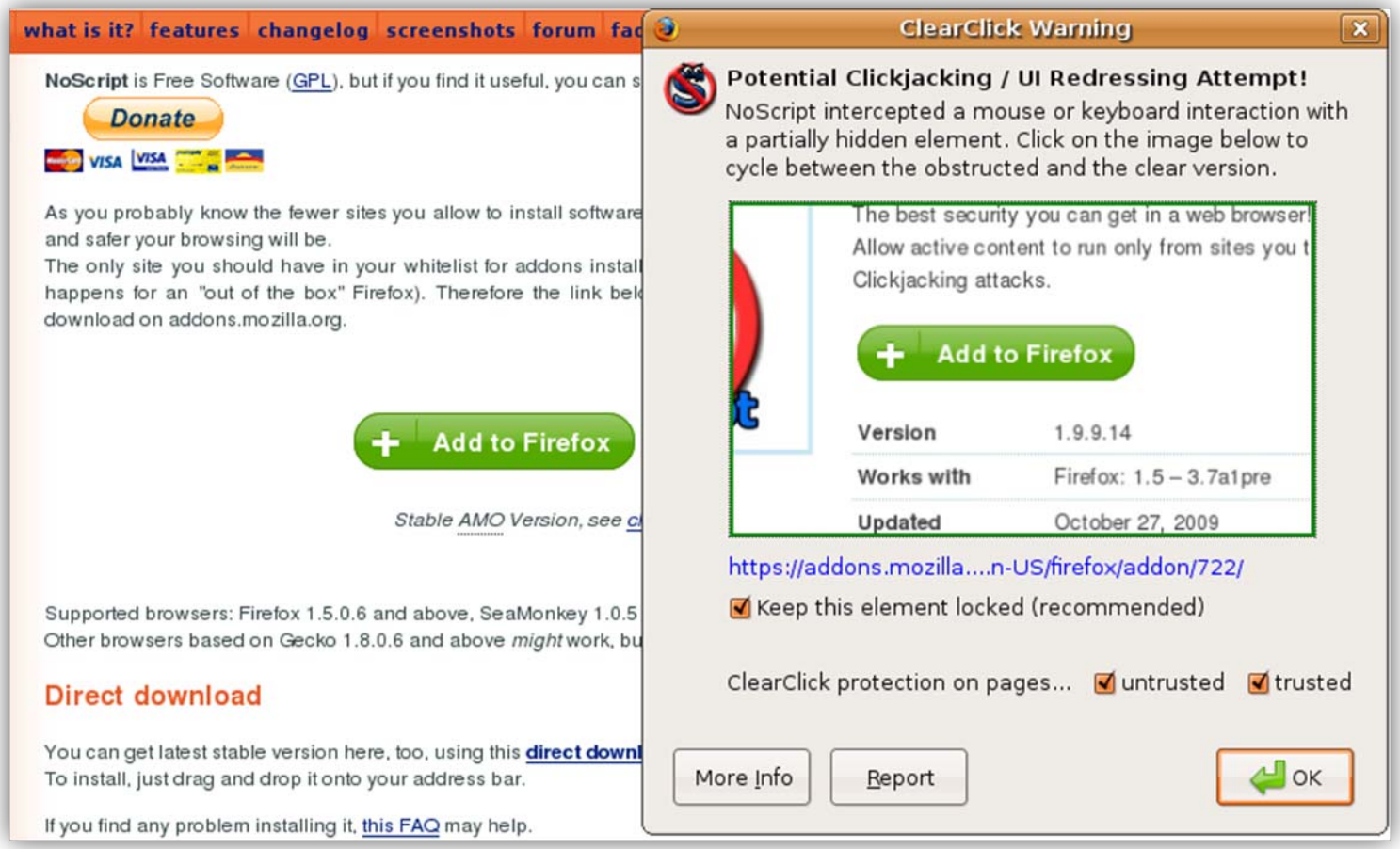
Application Boundary Enforcer (ABE)

Processing

- Processing order: SYSTEM, USER, Subscriptions, Site
- Each ruleset is processed top to bottom *until first match* (permissive exceptions should go higher than restrictions)
- On permissive or non-fatal outcome (`Accept`, `Anon`, `Sandbox`) processing resumes with next ruleset
- On restrictive fatal outcome (`Deny`) request is cancelled and processing aborted

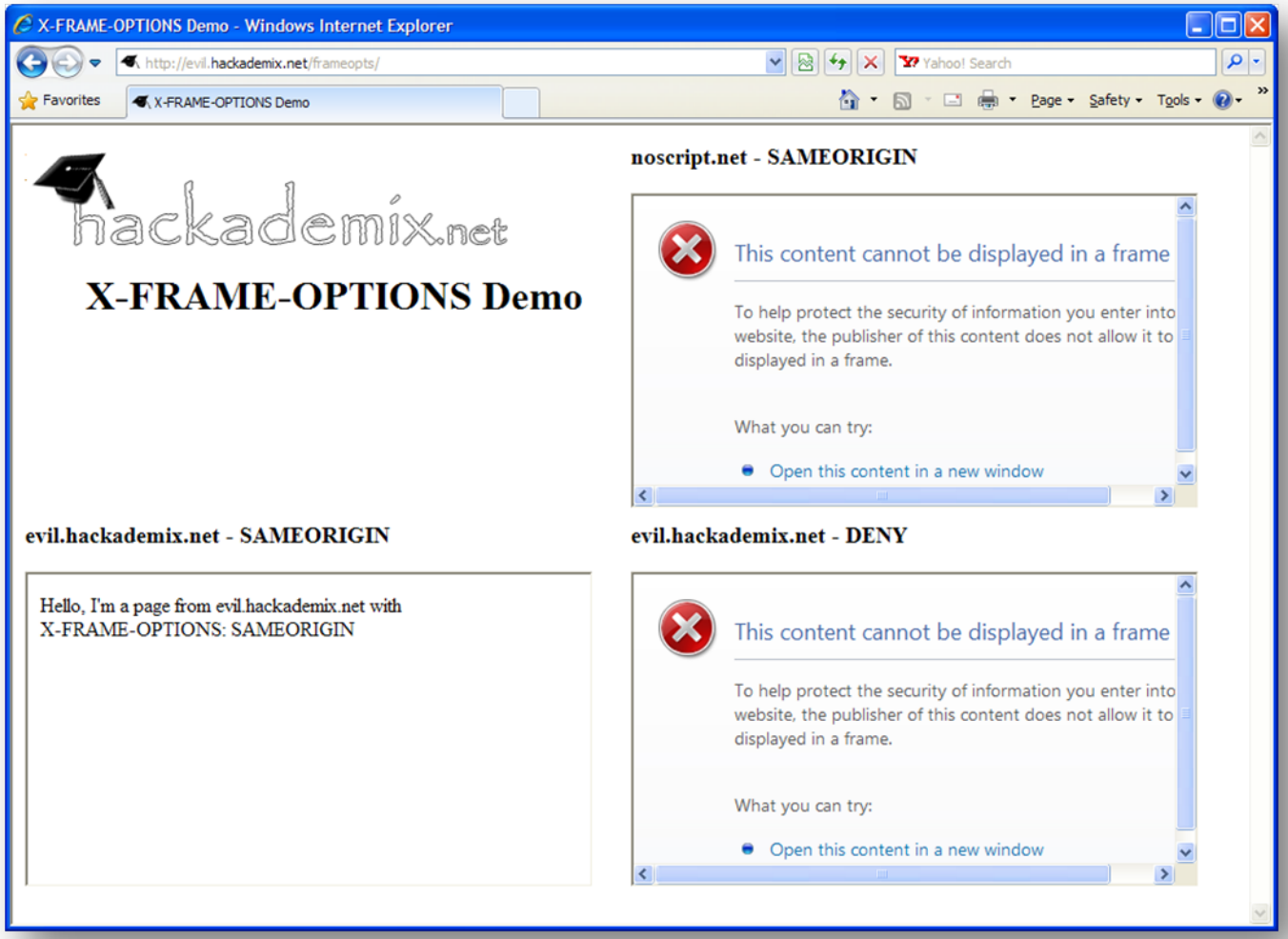
Clickjacking protection

ClearClick (NoScript)



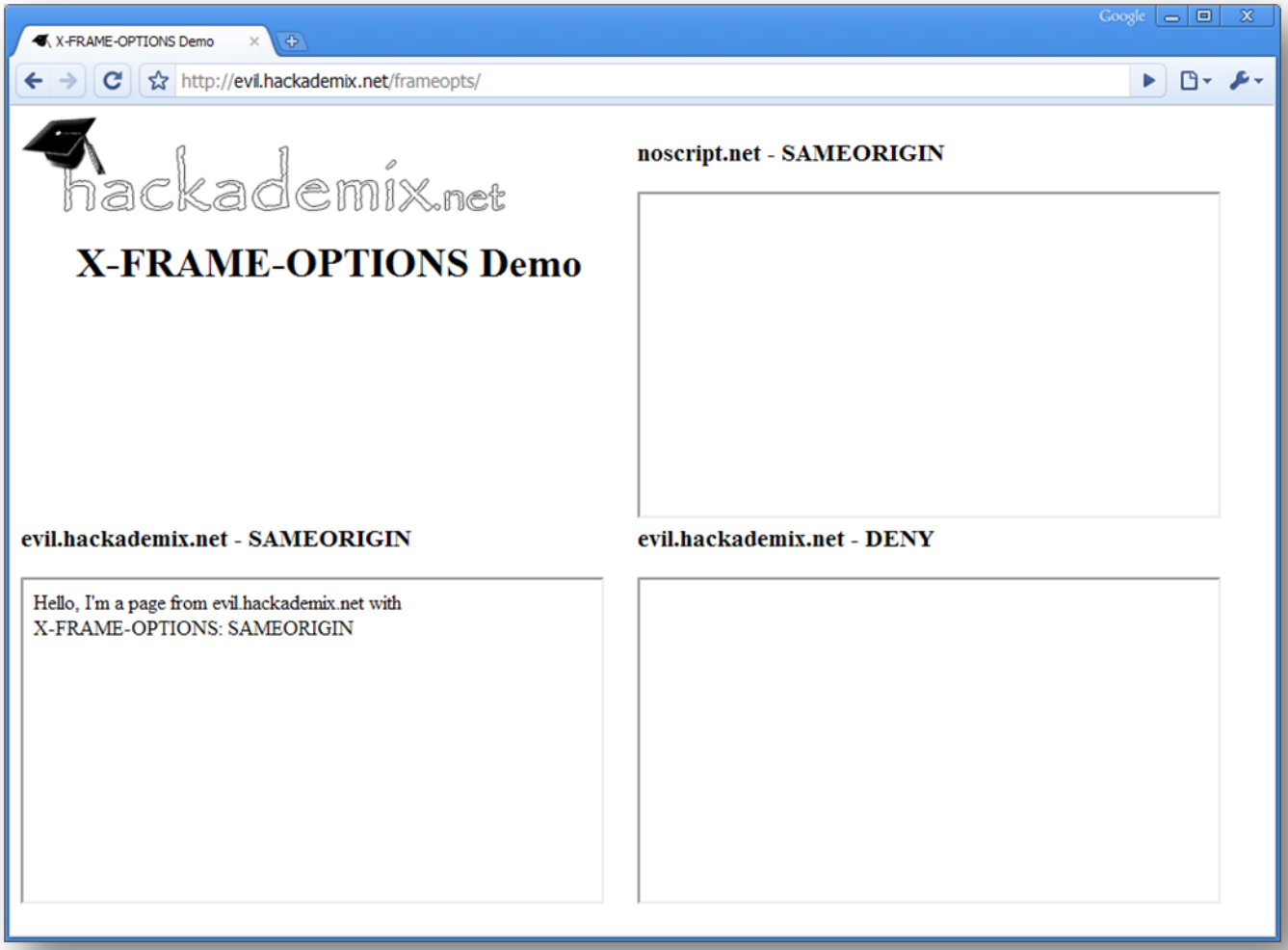
Clickjacking Protection

X-Frame-Options (MSIE)



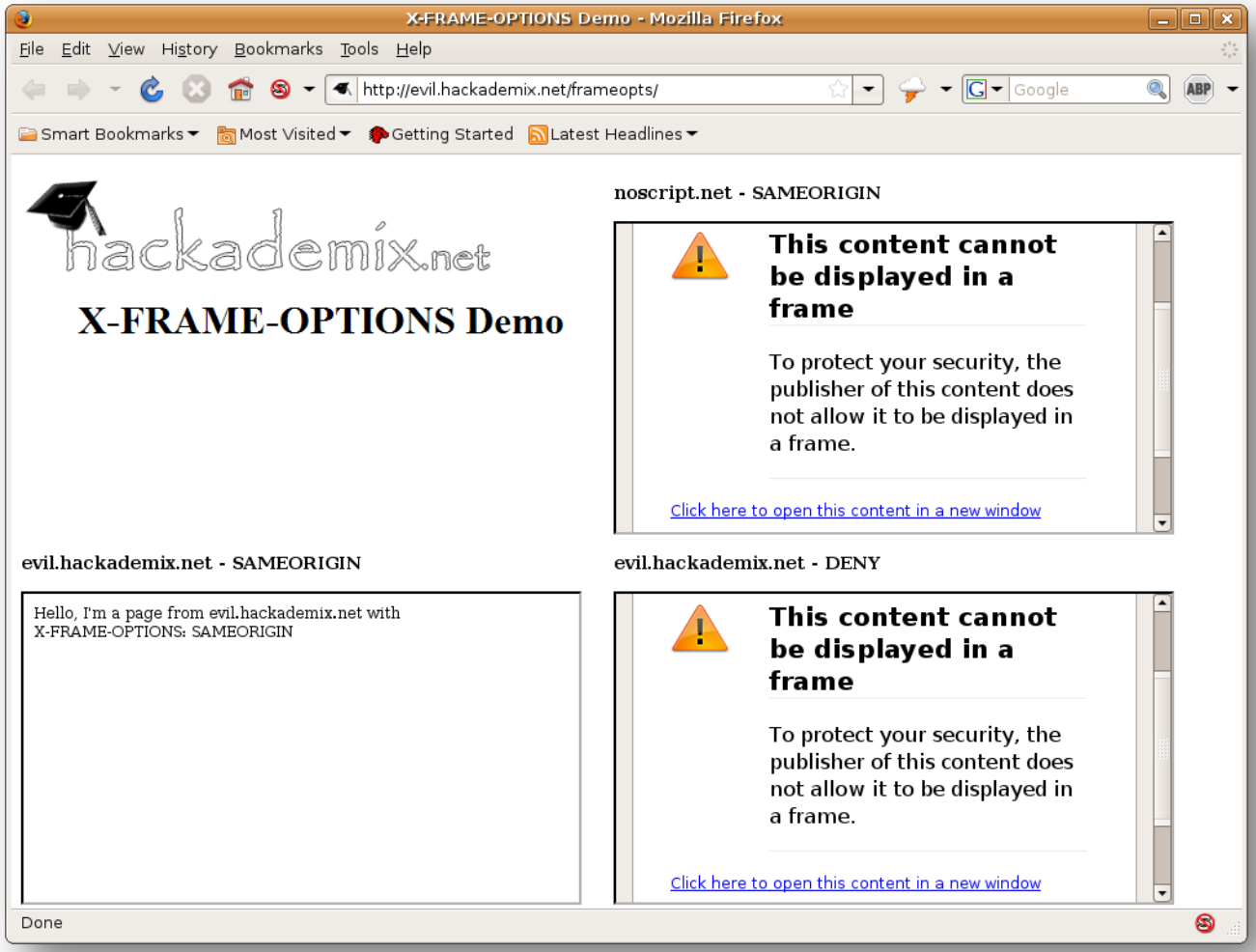
Clickjacking Protection

X-Frame-Options (Chrome)



Clickjacking Protection

X-Frame-Options (NoScript)



Clickjacking Protection

Other

- CSP's `frame-ancestors` directive (next Firefox)
- ABE's SUB action (NoScript)
- JavaScript framebusting

Developer?

Deploy X-Frame-Options + JavaScript framebusting (caveat)

User?

Use NoScript

References

NoScript, http://noscript.net

CSP, https://wiki.mozilla.org/Security/CSP/Spec

ABE, http://noscript.net/abe

InformAction, http://www.informaction.com

Giorgio Maone, http://maone.net