

Secure Coding. Practical steps to defend your web apps.

Copyright SANS Institute
Author Retains Full Rights

This paper is from the SANS Software Security site. Reposting is not permitted without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Defending Web Applications Security Essentials (DEV522)"
at <http://software-security.sans.org><http://software-security.sans.org/events/>

Protecting applications against Clickjacking with F5 LTM

GIAC (GWEB) Gold Certification

Author: Michael Nepomnyashy, mike.nepomny@gmail.com

Advisor: Johannes B. Ullrich, Ph.D.

Accepted: October 31th 2013

Abstract

Clickjacking is a web framing attack that uses iframes to hijack a user's web session. It is a powerful hacking technique that poses a threat to many types of web applications. The Information Security Organization of ACC Corporation decided to deploy centralized protection against clickjacking for hosted applications. The implementation of an anti-clickjacking solution can be quite challenging in a large scale hosting organization with over 70 applications that often frame each other. This paper describes a dynamic HTTP headers approach that protects hosted applications without breaking existing web framing relationship between webpages.

1. Introduction

OWASP defines clickjacking as “...when an assailant uses multiple transparent or opaque layers to trick a user into clicking on a button or link on another page when they were intending to click on the top level page. Thus, the attacker is "hijacking" clicks meant for their page and routing them to other another page, most likely owned by another application, domain, or both.” (Maas, 2013).

Facebook “likejacking” is one of the simplest clickjacking attack. The goal of “likejacking” is to trick unsuspecting Facebook user to “like” certain posts or pictures by routing their clicks to Facebook “like button”. Lets take this scenario:

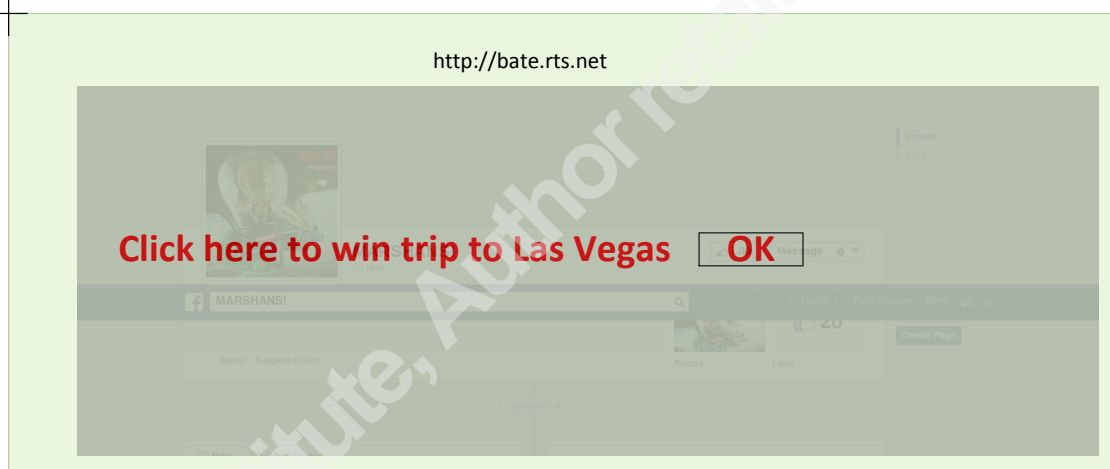
1. Attacker creates the “bait” page. The purpose of this page is to entice interest in victims and make them click on this page (Figure 1).
2. The targeted Facebook page with the questionable post is added as a frame on top of the “bait” page. The opacity is set to 0.0, making the Facebook page completely transparent (invisible) (Figure 2).
3. The “like” button on the invisible Facebook page is aligned with the “OK” button on the “bait” page. Every click on the “OK” button hits the “like” button on the targeted Facebook page (Figure 3).



Figure 1



Picture 2



Picture 3

So far, clickjacking attacks have been limited to social sites such as Twitter and Facebook (Stone, 2010). Next generation techniques such as text field injection, content extraction, HTML source extraction and forced drag-and-drop are considered possible (Stone, 2010). Build upon clickjacking, these attacks represent serious threats to many types of web applications.

The Information Security Organization of ACC Corporation decided to analyze different methods of mitigating clickjacking vulnerabilities and to implement the most effective one.

2. Effective Frame Busting

2.1. X-Frame-Options

A study of frame busting practices for the Alexa Top-500 sites showed that all techniques can be circumvented in one way or another (Rydstedt, Bursztein, Boneh, & Jackson, 2010). The same study recommended “.. a JavaScript-based defense to use until browser support for a solution such as X-FRAME-OPTIONS is widely deployed”.

The HTTP Response header X-Frame-Options is used to control whether or not a browser should be allowed to render a page in a frame or an iframe (Shahar, 2013).

There are two basic options:

DENY – The page cannot be displayed in a frame

SAMEORIGIN – The page can only be displayed in a frame on the same origin as the page itself.

A study which analyzed security headers of the top 1,000,000 websites reported that on March 10, 2013 the X-Frame-Options headers was the most popular of the security headers (Dawson, 2013)(Figure 4). The same study reported that SAMEORIGIN is by far the most common setting, followed by DENY (Figure 5). SAMEORIGIN ensures a good balance between protection from “clickjacking” and web-design flexibility.



Figure 4

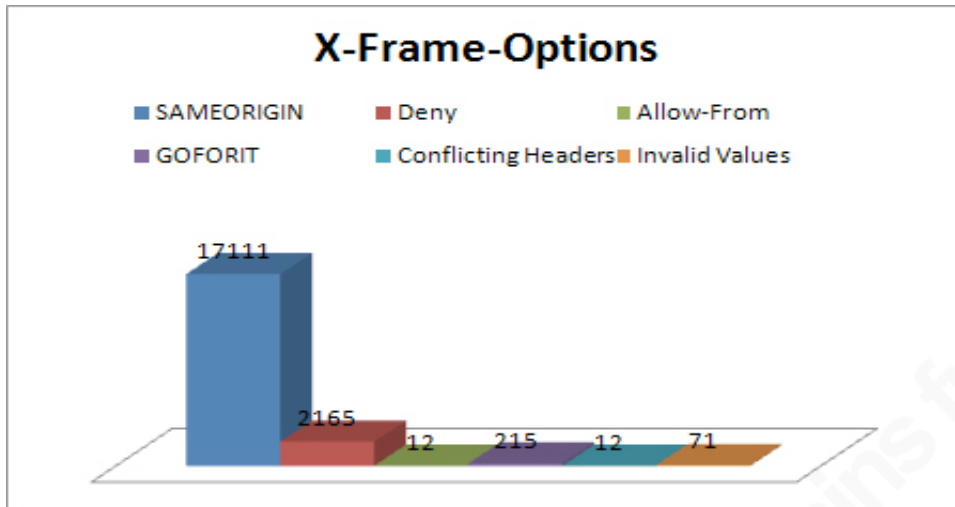


Figure 5

The following browsers support the X-Frame-Options headers (Shahar, 2013).

Browser	Chrome	FireFox	IE	Opera	Safari
Version	4.1.249.1042	3.6.9 (1.9.2.9)	8	10.5	4.0

The X-Frame-Options header can be set by the web application, a web server, or a network appliance. We will focus on the process of setting the “X-Frame-Options” header using BIG-IP LTM from F5 (Local Traffic Manager). This device features network-side scripting iRules (Pruitt). It takes one line of TCL based iRule to set the X-Frame-Options header for every response:

```
when HTTP_RESPONSE { HTTP::header insert "X-FRAME-OPTIONS" "SAMEORIGIN") }
```

This solution is sufficient for most applications but not for all. ACC engineers deployed the “framebusting” iRule as a pilot for the user account management application <https://usersecure.acc.com> and discovered that many ACC hosted applications are framing <https://usersecure.acc.com>. Using the “SAMEORIGIN” value broke existing functionality.

A typical multi-national corporation such as ACC, hosts 70 – 80 web applications. Many of these applications randomly frame each other, making DENY and SAMEORIGIN inappropriate values for the X-Frame-Options header.

Author Name, email@address

2.2. ALLOW-FROM X-Frame Options Value

Fortunately, X-Frame-Options can have a third value:

ALLOW-FROM *uri* - The page can only be displayed in a frame on the specified origin

The following browsers support the ALLOW-FROM directive:

Browser	Chrome	FireFox	IE	Opera	Safari
Version	Not supported	18.0	8	Not Supported	Not supported

The "ALLOW-FROM" directive supports only one URI. It does not support a list of URIs or wildcard characters. This restriction prevents a protected application from being framed by more than one URI, unless "ALLOW-FROM" can be changed dynamically.

In cases when the web application wants to allow more than one URI to frame its content, the following design can be used (D. Ross, 2013):

1. If web application A wants to render the requested content inside a frame, it provides its own origin information via a query string parameter to the web application B serving the to-be-framed content.
2. The application B verifies that the origin meets its criteria so the page can be allowed to be framed by the application A. The verification may happen via a search of a white-list of trusted URIs that allowed framing the application B content.
3. After successfully completing verification, the application B returns the URI in X-FRAME-OPTIONS: ALLOW-FROM header.
4. If a browser supports ALLOW-FROM directive, it enforces the X-FRAME-OPTIONS: ALLOW-FROM header that was set in step #3.

ACC engineers implemented the described scenario by writing a "framebuster" iRule running on an F5 LTM device v11.2.1. The following definitions are used in the "framebuster" iRule:

- outerframe - a page that wants to render the content inside a frame (D. Ross, 2013)
- innerframe – the to-be-framed content

Author Name, email@address

The origin of the outerframe page is set by the outerframe application via a query string parameter or session cookie. The outerframe value is a key to a hash table containing origin URIs. Table 1 summarizes “framebuster” iRule actions.

HTTP Request		HTTP Response	
outerframe parameter	outerframe cookie	X-Frame-Options	outerframe cookie
null	Null	sameorigin	null
a	Null	https://a.acc.com	a
null	b	https://b.acc.com	null
a	b	https://a.acc.com	a

Table 1

“framebuster” iRule Truth Table

An HTTP request can contain an outerframe parameter, outerframe cookie or both. Depending on this combination, the “framebuster” iRule sets the X-Frame-Options header with different values. An outerframe parameter takes precedence over an outerframe cookie. Same iRule also sets an outerframe cookie in response.

F5 offers an Application Security Manager™ (ASM) product. ASM is a Web Application Firewall. Starting with version 11.3, ASM can be configured to set X-Frame-Options header. Currently this feature is limited to one static URI.

2.3. The Outerframe Parameter Forgery

The solution presented above is immune to an outerframe query string parameter forgery attempts. Listed below are two cases:

1. In the first case a malicious website www.evil.net will load a page <https://a.acc.com> from the ACC website inside a frame and set a query string outerframe parameter *outerframe=www.evil.net*
2. The “framebuster” iRule performs search against internal hash table containing origin URIs.
3. The search returns nothing and iRule will set header *X-Frame-Options=SAMEORIGIN*
4. A browser displays the message “*This content cannot be displayed in a frame*” and attack will fail (Figure 6).

Author Name, email@address



Figure 6

A second case of an outerframe parameter forgery attempt can look like this:

1. A malicious website `www.evil.net` will load a page `a.acc.com` from the ACC website inside a frame and set a query string parameter `outerframe=b`
2. Application B is on the list of applications that can frame web application A. The "framebuster" iRule performs lookup and fetches the URI of application B.
3. The same iRule will set header `X-Frame-Options= ALLOW-FROM https://b.acc.com`
4. A browser sees the mismatch between outerframe origin information (`https://www.evil.net`) and URI `https://b.acc.com` set by the application A iRule in X-Frame-Options header and displays the message *"This content cannot be displayed in a frame"* (Figure 7).

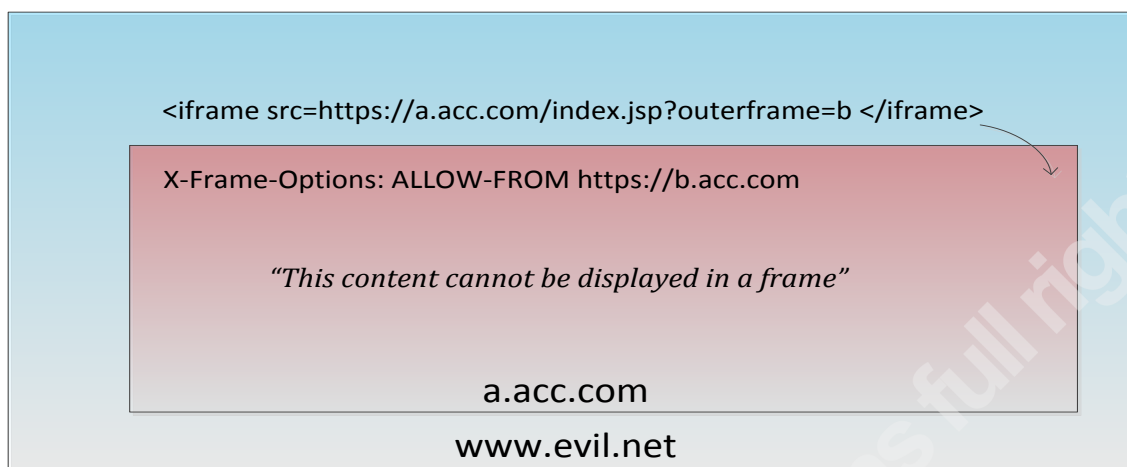


Figure 7

2.4. Nested Frames

A majority of ACC web applications can be categorized as either – innerframe or outerframe. There are a small number of applications that can be both, because they render content in a frame while simultaneously serve to-be-framed content. In this case we are dealing with nested frames. Figure 8 illustrates this relationship.

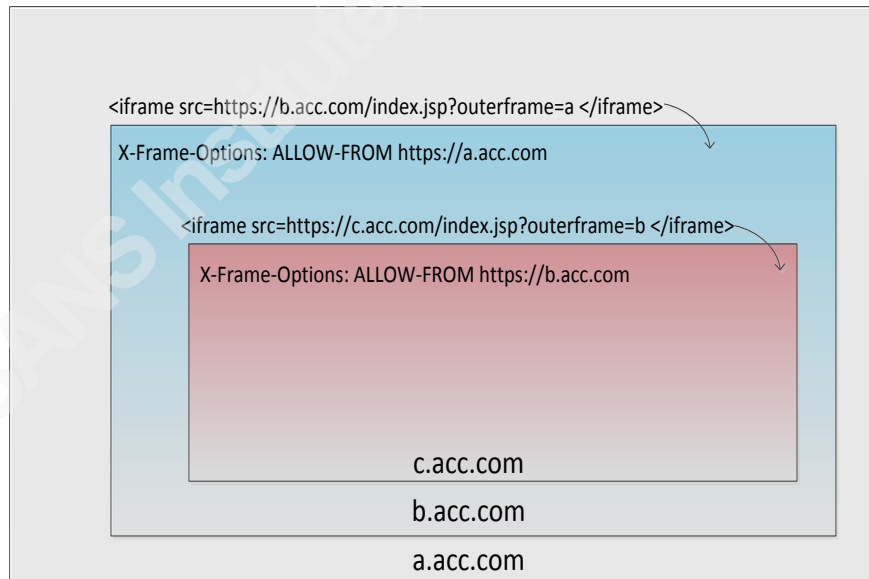


Figure 8

Application A frames application B, which in turn frames application C. If the framebusting iRule is set on B and C URIs the following will happen:

Author Name, email@address

1. Application A renders the content of application B in a frame with query string parameter *outerframe=a*.
2. Application B serves the content and the “framebuster” iRule sets the header
X-Frame-Options = ALLOW-FROM https://a.acc.com and browser displays B content in the frame
3. Application B in turn frames the content of application C with query string parameter *outerframe=b*
4. Application C serves a content and “framebuster” iRule sets the header to
X-Frame-Options = ALLOW-FROM https://b.acc.com
5. Browser sees the mismatch between top-frame origin information (*https://a.acc.com*) and URI *https://b.acc.com* set by the application C iRule in X-Frame-Options header and displays the message “*This content cannot be displayed in a frame*” .

One way to resolve this problem is to remove the framebusting iRule from web application C. In this case all nested frames will be presented, but application C will be vulnerable to clickjacking.

Applications B and C can be protected by enhancing application B. Application B will determine whether it is running in a frame. If it is, application B will not set an *outerframe* parameter or an *outerframe* cookie while rendering application C content in a frame. In this case, application C iRule will set the X-Frame-Options header based on the *outerframe* cookie set by the top application A. The browser will then have no difficulty displaying content C in a frame. Figure 9 illustrates that.



Figure 9

It is not trivial for application B to determine if it is running in a frame, nothing in the requests indicates that. The outerframe session cookie can still be used as a marker that application B is already framed. However this is not 100% reliable. An outerframe cookie could have been set by application D, for example, during previous session. The most reliable approach is to use a small snippet of JavaScript code to determine if application B is framed. This check will take place at the beginning of the session, and a session variable will be set as a reminder that application B content is rendered in a frame.

3. Conclusion

The framebusting technique described in this paper was successfully implemented at the ACC production environment for half-a-dozen applications. Security engineering was spearheading the deployment of “framebuster” iRule for selected applications front-ended by F5 BIG-IP™ Load Balancer. The process of setting and configuring the iRule does not involve an application development team. As a result the clickjacking protection for ACC applications is centralized, can be easily deployed and requires only minor application code change.

4. References

- Rydstedt, G., Bursztein, E., Boneh, D., & Jackson, C. (2010). Busting frame busting: a study of clickjacking vulnerabilities at popular sites. in IEEE Oakland Web 2.0 Security and Privacy (W2SP 2010).
- Maas, T. (2013, April). Clickjacking. Retrieved from www.owasp.org:
<https://www.owasp.org/index.php/Clickjacking>
- Pruitt, J. (n.d.). Introduction to iRules. Retrieved from DevCentral:
<https://devcentral.f5.com/tech-tips/articles/irules-101-01-introduction-to-irules>
- D. Ross, T. G. (2013, July 15). HTTP Header Field X-Frame-Options. Retrieved from IETF Tools: <http://tools.ietf.org/html/draft-ietf-websec-x-frame-options-05>
- Dawson, I. (2013, March). Security Headers on the Top 1,000,000 Websites: March 2013 Report. Retrieved from VERACODE:
<http://www.veracode.com/blog/2013/03/security-headers-on-the-top-1000000-websites-march-2013-report/>
- Stone, P. (2010, April 14). Next Generation Clickjacking. Retrieved from CONTEXT Information Security: http://www.contextis.co.uk/files/Context-Clickjacking_white_paper.pdf
- Shahar, R. (2013, August 26). *The X-Frame-Options response header*. Retrieved from Mozilla Developer Network: <https://developer.mozilla.org/en-US/docs/HTTP/X-Frame-Options>

Author Name, email@address

Upcoming SANS App Sec Training

Click Here to
{Register NOW!}

SANS Austin Winter 2017	Austin, TX	Dec 04, 2017 - Dec 09, 2017	Live Event
SANS Cyber Defense Initiative 2017	Washington, DC	Dec 12, 2017 - Dec 19, 2017	Live Event
SANS Security East 2018	New Orleans, LA	Jan 08, 2018 - Jan 13, 2018	Live Event
SANS Amsterdam January 2018	Amsterdam, Netherlands	Jan 15, 2018 - Jan 20, 2018	Live Event
Community SANS San Jose DEV534	San Jose, CA	Jan 29, 2018 - Jan 30, 2018	Community SANS
Community SANS Indianapolis DEV534	Indianapolis, IN	Feb 05, 2018 - Feb 06, 2018	Community SANS
Cloud Security Summit & Training 2018	San Diego, CA	Feb 19, 2018 - Feb 26, 2018	Live Event
Communtiy SANS Seattle DEV534	Seattle, WA	Feb 26, 2018 - Feb 27, 2018	Community SANS
San Francisco Spring 2018 - DEV522: Defending Web Applications Security Essentials	San Francisco, CA	Mar 12, 2018 - Apr 17, 2018	vLive
SANS San Francisco Spring 2018	San Francisco, CA	Mar 12, 2018 - Mar 17, 2018	Live Event
SANS Northern VA Spring - Tysons 2018	McLean, VA	Mar 17, 2018 - Mar 24, 2018	Live Event
SANS 2018	Orlando, FL	Apr 03, 2018 - Apr 10, 2018	Live Event
RSA Conference 2018	San Francisco, CA	Apr 11, 2018 - Apr 16, 2018	Live Event
SANS Baltimore Spring 2018	Baltimore, MD	Apr 21, 2018 - Apr 28, 2018	Live Event
Community SANS New York DEV522	New York, NY	Apr 23, 2018 - Apr 28, 2018	Community SANS
SANS Security West 2018	San Diego, CA	May 11, 2018 - May 18, 2018	Live Event
SANS Northern VA Reston Spring 2018	Reston, VA	May 20, 2018 - May 25, 2018	Live Event
SANS OnDemand	Online	Anytime	Self Paced
SANS SelfStudy	Books & MP3s Only	Anytime	Self Paced