CSE 509, 2014 Fall
Kelong Wang, Yongxin Liu
November 17, 2014

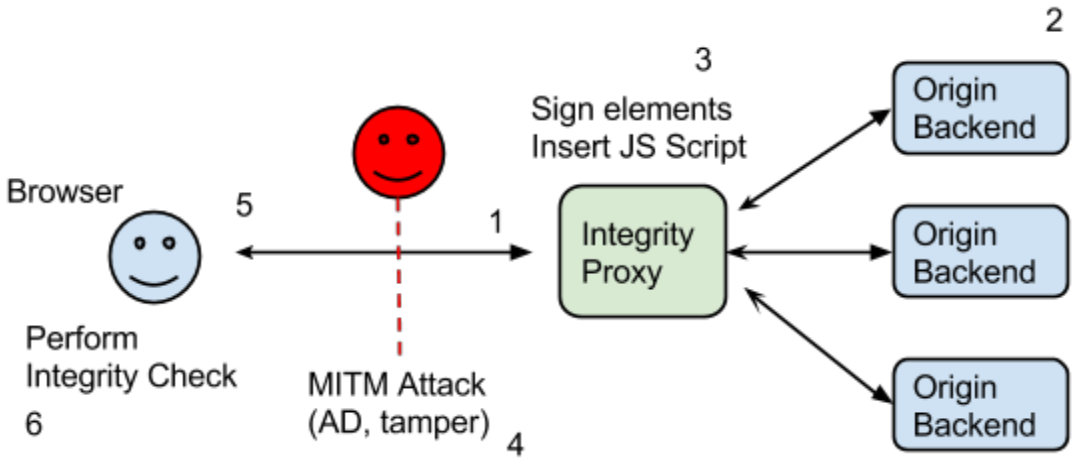
Web Integrity Reverse Proxy

Design

Overview

As we may experience it everyday surfing the Internet, the ISP or attackers on the traffic path can easily modify the HTTP payload to conduct subsequent attacks or simply add advertisements that do not belong to original website. Users and browsers can hardly distinguish between the genuine or fake one.

This project provides a practical way to protect the integrity of a web page (HTML). Our integrity proxy is deployed as front-end of the protected website (say origin.CNN.com). The CNN website administrator need to modify CNN.com DNS record to this proxy (via A record or CNAME record). So all the traffic to CNN.com will go through the proxy. The proxy will fetch original HTML from backend and add integrity tag (signature of content or simple pre-defined string) to each element. The proxy will also insert integrity check script to the HTML. When the browser receives the response, the script will remove additional elements and warn the user about the discrepancy.



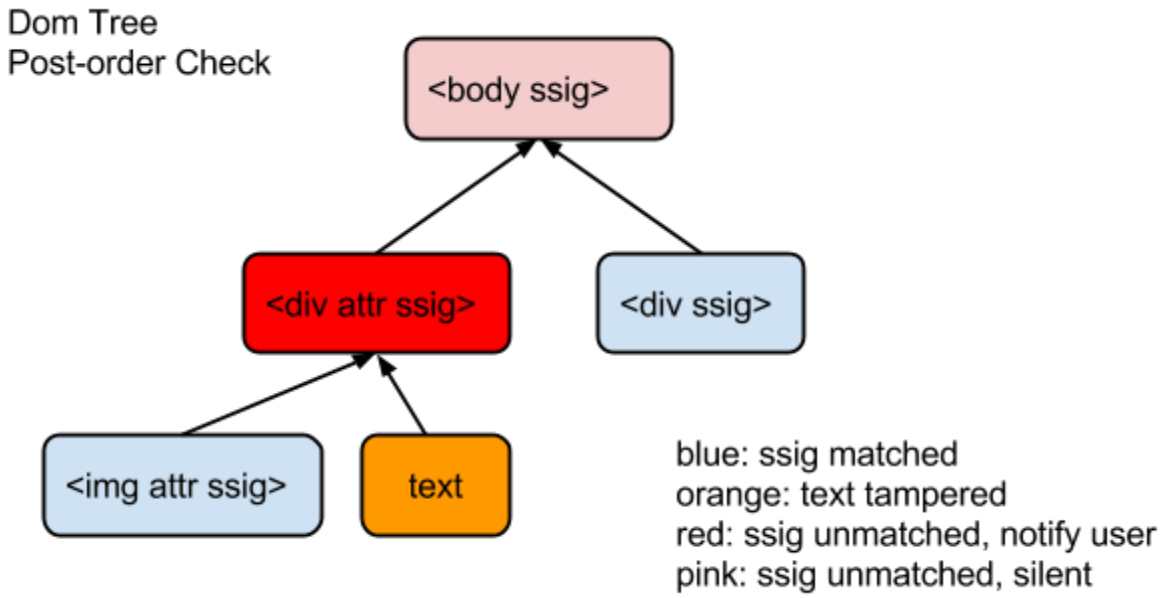
Steps

1. Browser issues a HTTP request to CNN.com which points to integrity proxy.
2. Integrity proxy forwards the request to (one of) backend and backend computes the original HTML to proxy.
3. Before sending response back to browser, the integrity proxy signs each element and insert JavaScript to the top of original HTML. It provides three options of signature.
    - Simple predefined string: `<div attr> → <div attr xtag='509'>`

- Symmetric hash: `<div attr> → <div attr md5>`
- Asymmetric signature: `<div attr> → <div attr ssig>`

The last option requires proxy to compute with its private key and include the public key into the script. We implement the first two options and assume the last option is available[1]. Note that the plaintext to be signed is the HTML inside each element. When we sign one element, we should make sure its content doesn't change later. So post-order (bottom to up) signing is used in the DOM tree.

4. The attackers modify the HTML.
5. The compromised response reaches the browser.
6. The integrity check script starts to sanitize the DOM tree from bottom to up.
    - Hide the elements without integrity attributes.
    - Compute signature of the HTML inside each element, for instance, a *div* element. If the order or attribute of children are altered, or the text is tampered, the signature will not match with the one in *div*'s attribute, and script will notify user in some ways.
    - The parent element of the compromised element (*body* in the figure below) can not pass the signature check either. But the script need not notify user again.



Attack, Defense and Assumption

- Element insertion
    - Normal attacker inserts an element without adding any integrity tags. It can be easily detected and removed.

[1] https://github.com/travist/jsencrypt

    - If we use symmetric hash like md5, attacker can insert an element with correct md5 attached. The attacker also needs to update all the parent nodes of the inserted element, otherwise it can be detected.
    - If we use asymmetric signing, attacker cannot forge a signature. But this approach will increase overhead on both proxy and client side.
- Body element modification / deletion
    - Same as element insertion, symmetric hash protection requires attacker to craft the element and attribute carefully.
    - If we use asymmetric signing, attackers can not forge the signature.
- Integrity script manipulation
    - Serious attacker will try to delete, modify or disable the integrity script. In this case, though the attacker cannot forge the signature, it is possible the script doesn't work correctly at all.
    - One weak workaround is to mix the integrity script with other general scripts. On one hand, it makes the attacker harder to figure out and modify. On the other hand, if attacker's modification can easily lead to incorrect behavior of page (e.g., users can hardly perform normal interaction with page or the page doesn't display well enough), we would say it is not a successful attack. At least users can tell it is not fully functional and would become more careful.
- Other
    - Another possible attack is to dynamically add elements which happens after integrity check. To prevent this attack, we need not only to eliminate the malicious script but also to stop the execution if elimination is not enough. We haven't investigated this prevention mechanism yet.

Usage

(Tested on Ubuntu 14.10)

Install

1. Install LuaJIT and OpenResty (a web server based on Nginx which supports Lua scripting)

```
$ apt-get install libpcre3-dev libssl-dev libluajit-5.1-dev luajit
$ wget http://openresty.org/download/ngx_openresty-1.7.4.1.tar.gz
$ tar zxvf ngx_openresty-1.7.4.1.tar.gz && cd ngx_openresty-1.7.4.1
$ ./configure --with-luajit && make && sudo make install
```

2. Install Lua DOM parser

```
$ git clone https://github.com/google/gumbo-parser.git
$ cd gumbo-parser && ./autogen.sh
$ ./configure && make && sudo make install

$ git clone https://github.com/craigbarnes/lua-gumbo.git
$ make LUA_PC=luajit
$ make check LUA=luajit
```

```
$ sudo make install LUA_PC=luajit
```

3. Prepare configurations

Nginx config and Lua scripts assume your working directory is `/root/integrity_proxy/`

```
$ unzip integrity_proxy
$ cd integrity_proxy
$ cp nginx.conf integrity_proxy.conf /usr/local/openresty/nginx/conf/
```

4. Run web server

```
(start) # /usr/local/openresty/nginx/sbin/nginx
(reload)# /usr/local/openresty/nginx/sbin/nginx -s reload
(stop)  # /usr/local/openresty/nginx/sbin/nginx -s stop
(logs)  # tail -f /usr/local/openresty/nginx/logs/{error,access}.log
```

Demo

Suppose your proxy's address is 127.0.0.1

```
# echo '127.0.0.1 cs origin.cs protected.cs attack1.cs defend1.cs
attack2.cs defend2.cs' >> /etc/hosts
```

1. View original page: http://origin.cs/
   - In demo, the virtual host *origin.cs* is a reverse proxy acting as original website.
   - In practice, this is the internal host as a backend that only integrity proxy is aware of.
   - Edit the last *server* in */usr/local/openresty/nginx/conf/integrity_proxy.conf* to choose target. Default is *https://www.cs.stonybrook.edu*.

2. View protected page: http://protected.cs/
   - The protected web page is the output of integrity proxy. The HTML source is expected to contain integrity check scripts and integrity attributes (*xtag* and *ssig*).
   - In practice, DNS record of the website should point to this host as a front-end.
   - So without attack, all users should see this page.

3. View compromised page with AD: http://attack1.cs/
   - This virtual host will fetch original page and insert AD without passing integrity proxy.
   - It is to emulate the normal MITM attack that happens everyday.
   - Large ADs are expected to display on both sides.

4. View sanitized page with AD: http://defend1.cs/
   - The page is inserted with AD by MITM attack initially. But due to the integrity script, the ADs without integrity attributes fade out.

5. View compromised page with tampered text: http://attack2.cs/
   - This virtual host will fetch original page and replace some texts without passing integrity proxy.

- It is to emulate more malicious attacks.
- By default (controlled in *tamper_content.lua*), ‘2014’ is replaced with ‘2015’ and ‘CS’ is replaced with ‘MATH’.

6. View sanitized page with tampered text: http://defend2.cs/
    - The page content is tampered by MITM attack. But due to the integrity script, the sections containing the tampered text are marked in red background. The computed signature is not matched with the one in integrity attribute. Script will print logs to developer console of browser.

7. View sanitized page with both attacks above: http://cs/

Summary

We come up with a conservative and practical way to protect the integrity of web page. The integrity reverse proxy can be easily deployed and integrated to the existing server architecture without modifying the original website. Though it is not perfect comparing with HTTPS, it introduces little overhead while providing configurable levels of integrity which is generally enough in many cases.