

Sommario

Introduzione	1
Capitolo 1 – Il clickjacking.....	2
1.1 Il concetto.....	2
1.2 Implementazioni.....	3
Capitolo 2 – Le estensioni di Mozilla Firefox	6
2.1 – Strumenti.....	6
2.1.1 – Mozilla Application Frame Work e Firefox Web Browser	6
2.1.2 – HTML, CSS e DOM.....	7
2.1.2 – XML e XUL.....	9
2.2 – Le estensioni a Mozilla Firefox	10
Capitolo 3 – La soluzione proposta: ClickJackThis!	12
3.1 – ClickJackThis! versione 0.1	12
3.1.1 – L’idea.....	12
3.1.2 – Implementazione.....	12
3.1.3 – Analisi del risultato	22
3.2 – ClickJackThis! versione 0.2	22
3.2.1 – L’idea.....	22
3.2.2 – L’implementazione	23
3.2.3 – Analisi del risultato	31
3.3 – ClickJackThis! versione 0.3	31
3.3.1 – L’idea.....	31
3.3.2 – Implementazione.....	32
3.3.3 – Analisi del risultato	37
Conclusioni	38
Bibliografia.....	40

Introduzione

Nella moderna società dell'informazione, internet ed in particolare il World Wide Web hanno acquisito fondamentale importanza, sociale e economica, grazie all'altissimo numero di utenti. Parallelamente, sono nati malintenzionati col fine di trarre guadagni illeciti, sfruttando le possibilità offerte dalla tecnologia. Uno dei più recenti attacchi informatici è il clickjacking, col quale è possibile dirottare, all'interno di una pagina web, i click dell'utente che, credendo di interagire con un oggetto di suo interesse (link, animazione Flash o altro) attiva in realtà un oggetto invisibile con scopi maligni. Obiettivo di questo elaborato è stato di trovare uno strumento per la protezione da un attacco basato su questo concetto. Si illustrano quindi alcune attuali implementazioni, basate principalmente su opportune regole CSS e tag HTML. Non essendo possibile modificare né tanto meno eliminare l'uso di tali tecnologie, si è ritenuto di dover lavorare lato utente. Mozilla Firefox è il browser scelto da oltre il 47% degli utenti di internet [4-h] e permette di essere esteso agevolmente al fine di svolgere attività personalizzate. Si è quindi sviluppata un'estensione che, a partire dall'unico elemento in possesso dell'utente, ovvero il codice della pagina web che si sta visitando, riesca a inferire la presenza di pericoli dovuti al clickjacking. Al fine di avere un prodotto efficiente nella rilevazione e che non inficiasse né prestazioni del browser né comfort di navigazione, si sono sviluppate tre versioni, aggiungendo funzionalità e migliorando le carenze.

Capitolo 1 – Il clickjacking

L'idea di collegamento ipertestuale è una delle basi su cui si appoggiano tutti i documenti presenti su Internet, ed è senza dubbio lo strumento principale che rende così fruibili i contenuti, dato che permette di raggiungere agevolmente pagine differenti, come fossero tutt'una. Si comprende allora come un malintenzionato che volesse preparare un attacco informatico, qualora riuscisse a trovare una vulnerabilità insita in questo concetto così fondamentale, potrebbe raggiungere un numero estremamente elevato di vittime. Una minaccia del genere esiste e risponde al nome di clickjacking. In questo capitolo si analizzerà il concetto di base e possibili implementazioni.

1.1 Il concetto

Il clickjacking è un attacco informatico utilizzato all'interno del World Wide Web per generare siti con l'obiettivo di frodare il navigatore, per violare la sua privacy o per carpire dati sensibili, di qualunque natura (economica, personale, industriale, ecc). Esso si basa sulla possibilità data dalla tecnologia CSS di poter sovrapporre due oggetti assegnando ad ognuno opportuni parametri di stile all'interno della stessa pagina. In particolare, è possibile creare all'interno di una pagina che riporti contenuti di interesse per il navigatore, un oggetto, sia esso un collegamento ipertestuale, un'immagine, un contenuto attivo come animazioni Flash o applet Java, sul quale chi visita la pagina ha interesse a cliccare. Sotto a questo, perfettamente allineato e nascosto, l'attaccante porrà un secondo oggetto, anche di tipo diverso dal primo, sul quale potrà essere attivo un collegamento ipertestuale oppure pulsanti cliccabili o altro. Il risultato è che l'utente, interessato ad accedere al primo oggetto, innocuo ma non attivo, viene invece "dirottato" sul secondo, invisibile, che potrà effettuare operazioni indesiderate.

1.2 Implementazioni

Una possibilità di far ciò è rappresentata dal componente iframe del linguaggio HTML [1]. Il suo scopo è quello di contenere un intero documento HTML e di poterlo inserire all'interno di una pagina allo stesso modo di immagini o altro. Il principale vantaggio è che è possibile applicare le proprietà di stile attraverso un comune foglio di stile. Ad esempio, sul sito <http://deisnet.deis.unibo.it/CJK/> si è creato un sito d'esempio nel quale viene visualizzata una pagina riportante varie scritte. Tre di esse appaiono come normali collegamenti ipertestuali, seguiti da una breve descrizione.



ClickJacking Attack Example

Welcome to Deisnet ClickJacking example page. The Following fake links will bring you around some click issues. The fake links are span elements set up through a negative z-index. The user believes to click on a link but she doesn't click on it, no javascript is needed, just CSS. Click on "E" of each "Example" Word. Enjoy your click.

Example 1 The browser opens a javascript stored in another page (in the backgrounded one)

Example 2 The browser performs a google search without user consensus (after that you need to reload the page to use Example 3)

Example 3 The browser accept a cookie from another page that the user doesn't see

This page seems to be a normal HTML page, no Javascript and Flash scripts are embedded, even a smart user may think that everything is legal but an attacker can still steal clicks doing whatever he wants. In other words the attacker tricks the user to click in something she cannot see by clicking in something she can see. This fraud is possible through three easy steps which every web developer should know. The first one is to load the malicious page on background through an "iframe" where he sets properly the CSS opacity value at 0. This makes the iframe content invisible. The next step is to create an artificial web page which fits perfectly with the underground one. If the created page doesn't fit properly on the backgrounded one, the mouse cursor might change going out to the iframe, alerting the user that something wrong is happening. As last step the attacker makes an HTML element that wants to get clicks, putting it on the hidden link and setting the CSS z-index property to be behind the invisible iframe.

©Copyright 2009
DEISNET
Via Venezia,52 - 47023 Cesena

L'analisi del codice sorgente, liberamente accessibile dall'utente (e questo sarà un punto fondamentale da cui partire per la progettazione di strumenti di difesa) rivela che la realtà è diversa. Si prenda ad esempio la prima linea. Essa non è altro che del semplice testo diviso in due parti cui vengono applicate, attraverso un file esterno, diverse regole di stile. Il codice HTML della sezione in esame:

```
<span class="clickjack1"> Example 1</span>
```

```
<span class="clickjack1description"> The browser opens a javascript  
stored in another page (in the backgrounded one)</span>
```

Si usano due classi definite in un CSS esterno. Esso contiene le seguenti:

```
.clickjack1{                                .clickjack1description{
background-color: white;                    background-color: white;
cursor:pointer;                            cursor:pointer;
color: blue;                               color: blue;
font-weight: bold;                        font-weight: bold;
font-size: 18px;                          font-size: 18px;
position: absolute;                        position: absolute;
top: 230px;                                top: 230px;
left: -10px;                              left: 100px;
z-index: -10;                             z-index: -10;
padding: 0px 0px 0px 0px;                padding: 0px 0px 0px 0px;
text-decoration: underline;                }
}
```

Si vede inoltre che all'interno di questa pagina vi è un iframe cui sono applicate ulteriori regole di stile:

```
<iframe                                id="attacksite" .backpage{
class="backpage"                        width="1000" opacity:0;
height="600"                             scrolling="no"   }
src="background.html" ></iframe>
```

Codice HTML per inserire l'iframe

Regole di stile applicate

Visualizzando con il browser la pagina background.html si ottiene il seguente risultato:

Background Page.

This the background page to exploit links. User beliefs to dik on other links whiles she is enforced to use these ones.

 Example 1 : normal link which executes javascript actions

 Example 2 : normal link to google page

 Example 3 : normal link to browser's cookies

©Copyright 2009
DEISNET
Via Venezia,52 - 47023 Cesena

Ora è chiaro cosa accade: questa pagina contiene pulsanti attivi che, quando cliccati, possono reindirizzare su un'altra pagina l'utente o attivare javascript. Il documento maligno viene inserito come iframe in quello principale e nascosta dall'attributo `opacity=0`; viene quindi sovrapposta, sfruttando opportunamente la flessibilità della tecnologia CSS, ai finti collegamenti. Essi in realtà sono semplici scritte, alle quali viene imposto che siano sottolineate e che il puntatore cambi allo stesso modo dei veri con la direttiva `cursor`. Per controllare che la sovrapposizione porti le scritte sopra ai pulsanti e non il viceversa, si usa l'attributo `z-index`. Il risultato finale è che cliccare su un innocuo collegamento porta ad attivare oggetti nascosti.

Le potenzialità di questo attacco sono direttamente collegate con quelle offerte dalle tecniche per il Web. Negli ultimi anni infatti, gli sviluppatori di siti, spinti dal crescente numero di navigatori, per offrire prodotti sempre più accattivanti e ricchi di contenuti sempre più multimediali, hanno richiesto sempre maggior controllo su ciò che viene mostrato all'utente, per nascondere gli aspetti più tecnici dell'argomento. Per questo che sono nati un numero incredibile di tecnologie; per citare le alcune: Java, JavaScript, Flash. Proprio su quest'ultima è basato un altro esempio di clickjacking, differente da quello presentato per realizzazione tecnica ma non per concetto.

Sul sito <http://guya.net/security/clickjacking/game.html> veniva presentato un gioco basato su JavaScript, nel quale l'utente doveva cliccare più velocemente possibile su pulsanti mobili. In realtà, nascosto sotto a questo gioco, c'era una animazione Flash, opportunamente creata per attivare la webcam dell'utente e registrare video. Tale video veniva poi mostrato all'utente, come monito a prestare maggiore attenzione a questo tipo di attacco. Oggi questo attacco non è più realizzabile in quanto Adobe, la casa produttrice di Flash, ha risolto questa vulnerabilità.

Si vede come il clickjacking, all'apparenza di semplice implementazione, possa essere efficacemente sfruttato per i più svariati scopi. È altrettanto chiaro non è pensabile di impedire, tecnologia per tecnologia, l'utilizzo di questo attacco, sia perché è sempre possibile per il malintenzionato usare le versioni precedenti, non sicure, sia perché può essere implementato con strumenti essenziali come il CSS o JavaScript, cui nessun utente o sviluppatore è disposto a rinunciare.

Capitolo 2 – Le estensioni di Mozilla Firefox

Ad oggi, Mozilla Firefox è il browser scelto da oltre il 47% dei navigatori [4-h]. È quindi lecito rivolgere ad esso le attenzioni per progettare uno strumento che protegga l'utilizzatore da attacchi clickjacking. La risposta deve essere cercata nelle estensioni, potente strumento messo a disposizione da Mozilla per aggiungere funzionalità al browser. Ovviamente, dovendo operare lato client, si avrà a disposizione la pagina web maligna. Si dovrà quindi avere a disposizione un algoritmo che la analizzi e ne riconosca la pericolosità. In questo capitolo si presenteranno brevemente gli strumenti necessari alla costruzione di una estensione.

2.1 – Strumenti

2.1.1 – Mozilla Application Frame Work e Firefox Web Browser

Nel gennaio del 1998 la Netscape Communications Corporation decise di rilasciare la maggior parte del codice sorgente del suo browser omonimo. L'applicazione open source che sarebbe nata avrebbe preso il nome di Mozilla e sarebbe stata gestita dalla Mozilla Foundation. Nel novembre dello stesso anno Netscape viene acquisito da AOL, Corporation che a causa dei modesti ricavi decide nel 2003 di terminare il famoso browser. Frattanto nel 2001, dopo un lungo sviluppo, nasce la prima versione stabile del browser Mozilla basato su una nuova layout engine, Gecko, ovvero il componente che si occupa della gestione delle nascenti tecnologie, molte ancora in bozza, quali CSS, XML, DOM ed altre [6]. Da allora Mozilla è progredita fortemente e allo stato attuale mette a disposizione agli sviluppatori un Application Frame Work estremamente completo [3-d] per progetti software di ogni tipo. Esso offre vari strumenti:

- Gecko: già menzionato, è la layout engine che si occupa di riconoscere i linguaggi con cui vengono scritte le pagine web e di generare ciò che viene visualizzato.
- Necko: estende Gecko offrendo le API necessarie alle applicazioni per operare dal livello di trasporto a quello di presentazione.
- XUL: applicazione dell'XML, è il linguaggio base per definire le interfacce utente.

- XPCOM: interfaccia ad oggetti per permettere agli sviluppatori di far comunicare componenti esterni scritti in linguaggi diversi (C++, Java, ecc) con l'ambiente Mozilla.
- XPConnect: componente che permette l'utilizzo di JavaScript come linguaggio d'elezione per le parti algoritmiche, fungendo da intermediario fra il linguaggio e XPCOM.
- XPInstall: componente che semplifica l'installazione e la gestione di temi ed estensioni nelle applicazioni.
- Molti altri componenti specifici per venire incontro alle più svariate esigenze (SQL, LDAP, ecc).

Il browser Firefox è basato proprio su questo frame work. Esso ha un'interfaccia grafica completamente scritta in linguaggio XUL il cui component principale è chiamato browser.xul ed è situato all'interno dell'archivio browser.jar, presente nella cartella di installazione del programma. Ogni estensione che voglia aggiungere elementi grafici all'interfaccia principale deve prevedere un file XUL che descriva un overlay, ovvero un insieme di elementi che si uniscono alla finestra del browser. La parte algoritmica viene invece di norma gestita attraverso JavaScript, potente linguaggio di scripting perfettamente in grado di interagire con i linguaggi tipici del web (HTML, CSS, DOM). Verrà in seguito analizzata in dettaglio la struttura di una estensione.

2.1.2 – HTML, CSS e DOM

Alla base di una qualsiasi pagina web c'è l'HyperText Markup Language [4-d]. Esso consente, attraverso opportune parole codificate, evidenziate fra parentesi angolari < >, di descrivere la struttura di un documento, inserendo oggetti quali testi, tabelle, immagini e quant'altro, il tutto in forma testuale, demandando al browser dell'utente la visualizzazione di quanto richiesto. Ovviamente nel tempo sono state standardizzate dall'organismo responsabile, il W3C, varie versioni di HTML, inserendo di volta in volta nuovi tag e nuove funzionalità; ad oggi, l'ultima versione è la 5. Tuttavia, stante l'esigenza di inter-operabilità con diversi formati, è stato parallelamente, sviluppato il XHTML, ovvero una riformulazione dello HTML standard adottando però le regole di

XML. In ogni caso, la struttura di un documento resta la stessa. Si ha una prima riga che indica la versione del linguaggio usato, ad esempio la versione 4:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

In seguito vengono aperti, annidati, sezioni della pagina, quali titolo, corpo principale, oggetti in esso contenuti ecc. Un esempio di semplice pagina potrebbe essere il seguente:

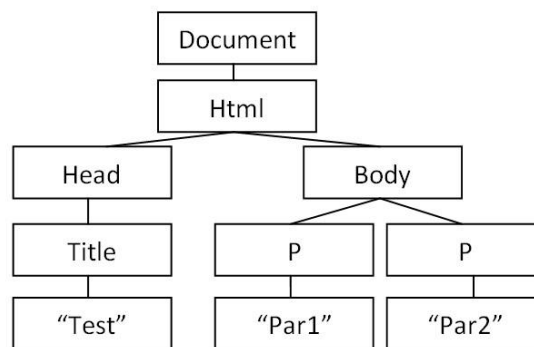
```
<html>
<head>
  <title>Test</title>
</head>
<body>
  <p>
    Par1
  </p>
  <p>
    Par2
  </p>
</body>
</html>
```

Lungo gli anni novanta, con la sempre maggiore necessità di funzionalità grafiche, ogni casa produttrice di browser aveva creato dei tag grafici proprietari, incompatibili fra di loro. Peraltro, erano da applicarsi elemento per elemento, aumentando ridondanza e dimensioni in byte delle pagine. La soluzione proposta dal W3C, divenuta standard, prende le mosse dal concetto di separare contenuti e presentazione, ed è il CSS, sigla per cascade style sheet [4-a]. Questa tecnologia permette di definire regole di grafiche univoche, standard per ogni browser, contenute in un file diverso dalla pagina HTML. Ciò viene fondamentalmente effettuato con tre metodi:

- selettori di tipo: applicano la regola a tutti gli elementi della pagina del tipo determinato, come ad esempio ad ogni tag <p>.
- classi: applicano la regola a tutti gli elementi della pagina che presentano la proprietà `class="nome_classe"`.

- identificatori: applicano la regola a quell'elemento della pagina che presenta la proprietà `id="nome_identificatore"`. Solo un elemento in tutta la pagina può corrispondere ad un identificatore.

È ovviamente impensabile di trattare questi documenti, sempre più complessi, in forma testuale, ovvero senza un modello che permetta di navigare tag e loro proprietà in modo razionale. Per risolvere anche questo problema, il W3C ha standardizzato un modello di rappresentazione ad albero del documento, il DOM, acronimo di document object model [4-b]. Con esso si può rappresentare la pagina d'esempio sopra riportata con un albero del tipo:



Questa rappresentazione permette al programmatore che si trova a dover trattare con il documento di poterlo navigare come un usuale albero, posto che il linguaggio utilizzato supporti le API del DOM; ad esempio, si può identificare il titolo del documento, in metalinguaggio, come `document.head.title`.

2.1.2 – XML e XUL

XML è un metalinguaggio marcatore [4-i], ovvero che serve a definire un meccanismo sintattico per estendere o controllare il significato di altri linguaggi marcatori. Con esso è possibile definire nuovi linguaggi marcatori, come ad esempio HTML, basati ovvero su tag che identificano oggetti e loro proprietà. XML è molto rigido sulla sintassi da seguire: i tag non possono iniziare con numeri o caratteri speciali e devono essere bilanciati, ovvero non sono consentiti errori di annidamento; è inoltre case sensitive. Per poter essere correttamente interpretato da un browser, un documento XML deve essere ben formato ovvero avere:

- prologo: è la prima istruzione che appare scritta nel documento e serve a identificare la versione usata
- elemento radice: è il nodo principale, chiamato che contiene tutti gli altri nodi del documento e deve essere unico
- tutti i tag devono essere bilanciati.

XUL è un linguaggio per la creazione di interfacce grafiche basato su XML [3-g]. Il generico progetto XUL è diviso in tre parti fondamentali: content, skin e locale; esse devono essere separate in tre cartelle divise ed omonime e possono contenere sottosezioni comunque complicate. In generale, la prima comprende tutti i componenti grafici quali overlay, finestre di dialogo, pulsanti, menu ecc. e le azioni JavaScript [4-e] ad essi associati; la seconda, le regole di stile in formato CSS da applicare ai componenti grafici nonché icone ed eventuali immagini usate; la terza infine, le stringhe di testo che, all'interno del progetto, vengono usate per interagire con l'utente, ognuna di essa scritta in diverse lingue. Tali stringhe devono essere definite come entità dtd [4-c]. Questo ultimo punto ha lo scopo di generare software localizzabile, ovvero che possa rivolgersi all'utente nell'idioma opportuno, in modo efficiente, ovvero evitando al programmatore di cambiare le stringhe all'interno del codice, rimanendo fedeli agli standard W3C. In XUL infine, ogni file può essere referenziato con un normale URL di tipo http. Tuttavia, per evitare il problema di percorsi differenti da sistema a sistema, a seconda del tipo di installazione effettuata ad esempio, si è scelto di utilizzare uno speciale URI, detto chrome. La struttura di un URI di tipo chrome è la seguente:

```
chrome://<package>/<part>/<file>
```

dove <package> è il nome del progetto, <part> può essere content, skin o locale o eventuali loro sottosezioni, mentre <file> indica il file che si sta indicando.

2.2 – Le estensioni a Mozilla Firefox

Un'estensione Firefox è un package contenente un progetto XUL [3-a]. È quindi necessario procedere alla creazione di una cartella principale avente un nome identificativo dell'estensione. Tale nome può essere o il nome scelto per titolare il

prodotto o un GUID, acronimo di Globally Unique Identifier, identificatore unico globale, ovvero un numero pseudo casuale che identifica univocamente il prodotto. Esso può essere generato tramite opportuni script (pubblicamente reperibili on-line [2]) che rispettino la sintassi Mozilla. All'interno della cartella principale, dovrà essere creata una cartella denominata "chrome"; essa è il punto di partenza dell'URI chrome, e conterrà la parte XUL. Sono poi richiesti due file ulteriori, da affiancarsi alla cartella chrome: "chrome.manifest" e "install.rdf". Il primo descrive la struttura del package, ovvero le strutture delle parti content, skin e locale con la seguente sintassi:

```
<part> <package_name> <chrome_uri>
```

Dove <part> indica la sezione del progetto di cui si rende noto il percorso, <package_name> l'identificativo dell'applicazione e <chrome_uri> l'effettivo percorso in formato chrome. Il file manifest deve inoltre indicare quali file dell'estensione devono essere considerati overlay, ovvero aggiunte, e di quali parti del browser, con la seguente sintassi:

```
overlay <browser_part> <extension_part>
```

con i percorsi in formato chrome. Il file "install.rdf" contiene informazioni di installazione e di aggiornamento dell'estensione. Come indica il nome, esso va redatto secondo lo standard rdf del W3C [4-f]. Nel caso infine di estensioni che prevedano una serie di parametri che ne regolino l'esecuzione, detti comunemente preferenze, si dovrà procedere nella cartella principale di un'ulteriore cartella chiamata "defaults" contenente a sua volta una cartella "preferences" il cui unico file dovrà essere "default.js", avente opportuni comandi JavaScript per la dichiarazione e gestione. Una volta creata questa struttura e le parti di software, si procede alla creazione di un archivio zip del tutto e si cambia l'estensione del file ottenuto in .xpi, ottenendo così un oggetto gestibile da Firefox. Si veda le implementazioni proposte al prossimo capitolo per esempi specifici.

Capitolo 3 – La soluzione proposta: ClickJackThis!

Come evidenziato ai capitoli precedenti, ciò che si ha disposizione sono i file html e css delle pagine web che si stanno visitando. Sarà necessario quindi prevedere un algoritmo che, analizzando il codice in modo efficace, possa rilevare la presenza di eventuali rischi. Lo sviluppo del progetto ha seguito la seguente logica: si è pensato ad una soluzione, la si è implementata e si sono valutati pro e contro; si è quindi cercato di evolverla salvando gli aspetti positivi ed eliminando quelli negativi, fino ad ottenere il risultato desiderato.

3.1 – ClickJackThis! versione 0.1

3.1.1 – L'idea

Come si è già evidenziato, si è subito ritenuto di dover rilevare, all'interno del codice della pagina, elementi di rischio, in base ai quali decidere se avvisare l'utente o meno della pericolosità del sito visitato. Si è quindi inizialmente definita "maligna" una pagina che contemporaneamente presentasse: regole CSS che imponessero z-index negativo, regole che rendessero invisibile un oggetto con opacity:0, presenza di iframes. L'analisi viene effettuata appena la pagina viene caricata o in una nuova finestra o in una nuova tab. Fatto ciò, si è pensato di interfacciarsi con l'utente attraverso un messaggio di allerta ed attraverso una piccola icona in basso a destra nella finestra principale del browser, in stile semaforo, ovvero che diventasse rossa per le pagine rilevate come pericolose e verde altrimenti.

3.1.2 – Implementazione

Si procederà ora all'analisi dei file componenti l'estensione. La struttura è la seguente:



Come si vede, si è generato un GUID attraverso lo script riportato in bibliografia, che in questo risulta {cb6d3ea9-ff46-4f80-aaab-a6293952d011}. Inoltre si è generato il file install.rdf; la prima riga indica la versione di xml usata:

```
<?xml version="1.0"?>
```

Si procede poi alla qualificazione del tag <RDF> mediante file esterni, standard W3C e Mozilla:

```
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:em="http://www.mozilla.org/2004/em-rdf#">
```

Si forniscono poi il GUID dell'applicazione, la sua versione e la sua tipologia (ovvero se è un tema grafico, un'estensione o altro; il codice per le estensioni è il 2):

```
<Description about="urn:mozilla:install-manifest">
  <em:id>{cb6d3ea9-ff46-4f80-aaab-a6293952d011}</em:id>
  <em:version>0.1</em:version>
  <em:type>2</em:type>
```

Si definisce poi quale applicazione Mozilla si intende estendere; trattandosi di Firefox, si indica il suo GUID e il range di versioni per le quali si è testato il corretto funzionamento:

```
<em:targetApplication>
  <Description>
    <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id>
    <em:minVersion>3.0</em:minVersion>
    <em:maxVersion>3.0.*</em:maxVersion>
  </Description>
```

```
</em:targetApplication>
```

Infine, si danno informazioni descrittive, che saranno presentate all'utente nel gestore delle estensioni. In questo caso, per semplicità, si sono riportate direttamente le stringhe di testo desiderate; si sarebbe potuto sfruttare le possibilità di localizzazione indicando un identificativo generico per la stringa, concretizzato da Firefox di volta in volta, basandosi sui file dtd presenti nella cartella locale.

```
<em:name>ClickJackThis!</em:name>
```

```
<em:description>Protects you against ClickJacking based attacks.</em:description>
```

```
<em:creator>Simone Colella</em:creator>
```

```
<em:homepageURL>http://deisnet.deis.unibo.it/</em:homepageURL>
```

```
</Description>
```

```
</RDF>
```

Il file `chrome.manifest` definisce i percorsi delle sezioni del progetto XUL, qui ridotte alle sole cartelle `content` e `skin`; indica inoltre che nel progetto è presente un `overlay` e che esso deve essere inteso come aggiunta al browser principale.

```
content ClickJackThis chrome/content/
skin ClickJackThis classic/1.0 chrome/skin/
overlay chrome://browser/content/browser.xul
chrome://ClickJackThis/content/ClickJackThis.xul
```

Realizzata questa parte, si è passato alla realizzazione dell'interfaccia grafica. Il file `clickjackthis.xul` estende la finestra del browser. Dichiarata la versione di xml, si procede ad includere un foglio di stile esterno.

```
<?xml version="1.0"?>
```

```
<?xml-stylesheet href="chrome://ClickJackThis/skin/ClickJackThis.css"
type="text/css"?>
```



Si passa quindi a qualificare il tag `overlay` attraverso un file esterno fornito da Mozilla, che più in generale definisce lo XUL:

```
<overlay id="ClickJackThis"
```

```
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
```


Si include poi un JavaScript esterno; si noti che all'atto dell'inclusione esso viene immediatamente eseguito.

```
<script type="application/x-javascript"
    src="chrome://ClickJackThis/content/ClickJackThis.js" />
```

Infine, si crea una status bar di tipo “a icona” [3-e]. Per rendere possibile il cambiamento di icona, si è usata una possibilità offerta da CSS2, i selettori [4-g] che permettono di parametrizzare un id in funzione di un attributo, in modo che le regole applicate ad uno stesso oggetto, identificato univocamente appunto dall'id, cambino in funzione del valore imposto esternamente a tale attributo. Si è quindi definito un attributo checkresult che può assumere due valori: “ok” o “alt”; al file CSS esterno è demandata la scelta dell'immagine alt.png oppure ok.png, rispettivamente un cerchio rosso  e verde .

```
<statusbar id="status-bar">

    <statusbarpanel id="clickjacking-statusbar-panel"
        class="statusbarpanel-iconic"
        checkresult="ok"
        tooltip="ClickJackThis!"
```

Al click sull'icona, si chiede di aprire una finestra di dialogo personalizzata riportante informazioni sull'estensione. Quindi si chiudono i tag rimasti aperti.

```
onclick="window.openDialog('chrome://ClickJackThis/content/about.xul',
    '', 'centerscreen,chrome,resizable=no');" />

</statusbar>

</overlay>
```

L'id parametrizzato della barra di stato è descritto nel file clickjackthis.css:

```
#clickjacking-statusbar-panel[checkresult="ok"] {
    list-style-image: url("chrome://ClickJackThis/skin/ok.png");
}
```

```
#clickjacking-statusbar-panel[checkresult="alt"] {
    list-style-image: url("chrome://ClickJackThis/skin/alt.png");
}
```

La finestra di dialogo è descritta dal file about.xul:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
```

Si dichiara che il file descrive una finestra di dialogo, avente titolo opportuno, la cui dimensione è dimensionata automaticamente per contenere tutti i suoi elementi.

```
<window class="dialog"
    title="ClickJackThis 0.1 - About" orient="vertical"
    autostretch="always" onload="sizeToContent()"

xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
```

Si crea un componente groupbox che serve a contenere varie etichette di testo, fra cui una contenente il link alla homepage del progetto. Si è anche predisposto un pulsante di chiusura.

```
    <groupbox align="center" orient="horizontal">
        <vbox>
            <text value="ClickJackThis!" style="font-weight: bold; font-size: x-large;"/>
            <text value="ver. 0.1"/>
            <separator class="thin"/>
            <text value="Created By:" style="font-weight: bold;"/>
            <text value="Simone Colella" />
            <separator class="thin"/>
            <text value="Home Page:" style="font-weight: bold;"/>
            <text value="http://deisnet.deis.unibo.it" class="url"
                onclick="window.open('http://deisnet.deis.unibo.it/');
window.close();"/>
            <separator class="thin"/>
        </vbox>
        <spring flex="1"/>
    </groupbox>
```

```

<hbox>
  <spacer flex="1"/>
  <button label="Exit" onclick="window.close();" />
</hbox>
</window>

```

Il risultato finale è il seguente:



La parte attiva dell'estensione è contenuta nello script `clickjackthis.js`. Si noti che esso viene immediatamente eseguito non appena il browser viene avviato, dato che viene incluso da un suo overlay. In JavaScript, specialmente quando usato nelle estensioni Firefox, l'architettura di riferimento è quella ad eventi [3-b]. Per cui si associa agli eventi "apertura di una pagina" e "chiusura di una pagina" due funzioni opportune.

```

window.addEventListener("load",function (){
gBrowser.addEventListener("load", OnPageLoad, true);},false);
window.addEventListener("pagehide", OnPageUnload, false);

```

Si memorizzano le pagine ritenute maligne in un array:

```
var riskyPages = new Array();
```

L'evento "apertura di una pagina" viene generato per ognuna delle seguenti situazioni: caricamento del documento principale, caricamento di ogni frame o iframe in esso eventualmente presenti, caricamento dell'eventuale favicon. Si deve verificare quindi che l'oggetto che ha generato l'evento sia un documento HTML e che sia la pagina principale e non un suo frame.

```

function OnPageLoad(event) {

  if (event.originalTarget instanceof HTMLDocument) {

    if (event.originalTarget.defaultView.frameElement)
      return;

```

Si associano poi agli eventi tipici della navigazione a schede ad opportune funzioni [3-c]: la chiusura di una scheda viene gestita dalla stessa che gestisce la chiusura di una pagina; mentre l'apertura e la selezione di una scheda vengono gestite da funzioni dedicate.

```
var container = gBrowser.tabContainer;
var container2 = gBrowser.mPanelContainer;
container.addEventListener("TabClose", OnPageUnload, false);
container.addEventListener("TabOpen", tabOpened, false);
container2.addEventListener("select", tabSelected, false);
```

È possibile avere un riferimento univoco per ogni pagina web che il browser ha aperto nella sessione corrente; esso è dato coerentemente da un campo dell'evento che scatena la chiamata della funzione. Ci si assicura quindi che questo riferimento sia quello della pagina principale e non ad un suo frame, quindi lo si passa ad una funzione che lo analizzi.

```
var doc = event.originalTarget;
    while (doc.defaultView.frameElement)
doc=doc.defaultView.frameElement.ownerDocument;
testPage(doc),
}
}
```

La funzione seguente gestisce la chiusura di una pagina (o di una scheda). Il suo compito è quello di confrontare il riferimento della pagina in chiusura e di verificare se esso sia quello a una pagina marcata come a rischio, per cui memorizzata nell'array. Nel caso, il riferimento viene eliminato dall'array e viene ripristinata l'icona verde.

```
function OnPageUnload(aEvent) {

    if (aEvent.originalTarget instanceof HTMLDocument) {

var doc = aEvent.originalTarget;
    for(i=0; i<riskyPages.length; i++){

        if(riskyPages[i]==doc){
            riskyPages.splice(i,1);
            ok();
        }
    }
}
```

```

                break;
            }
        }
    }
}

```

La seguente funzione è richiesta dal caso di apertura in background di una scheda, il che avviene cliccando con il tasto destro su un link e scegliendo l'opzione "apri in una nuova scheda". Nel caso, la pagina viene analizzata, cosa che non sarebbe avvenuta altrimenti.

```

function tabOpened(event) {
    if(gBrowser.selectedBrowser != event.target.linkedBrowser)
        testPage(event.target.linkedBrowser.contentDocument);
}

```

Nel caso di più schede aperte, selezionandone una si vuole che l'icona diventi rossa o verde a seconda che la pagina sia stata precedentemente ritenuta malevola. Allora si verifica che il riferimento alla pagina visualizzata sia presente nell'array e, in tal caso, l'icona passa al rosso, oppure qualora sia assente, al verde.

```

function tabSelected(event) {

    var doc =
gBrowser.getBrowserAtIndex(gBrowser.mTabContainer.selectedIndex).content
Document;

    for(i=0; i<riskyPages.length; i++){

        if(riskyPages[i]==doc) {
            alt();
            break;

        }else{
            ok();
        }
    }
}

```

Alla seguente funzione è demandata l'analisi della pagina. Se il risultato (booleano) dei controlli effettuati sui CSS e sulla presenza di iframe (fatti da funzioni esterne) è positivo,

si manda un avviso all'utente, si fa diventare rossa l'icona e si pone il riferimento del documento nell'array delle pagine maligne.

```
function testPage(doc) {  
  
    var maliciousStyle=testStyle(doc);  
    var iframesUsed=testIframes(doc);  
  
    if(maliciousStyle && iframesUsed){  
  
        alt();  
        alert("Possible ClickJacking Attempt!");  
        riskyPages.push(doc);  
    }  
  
}
```

Il controllo sui CSS avviene caricando in un array tutti i fogli di stile del documento, interni o esterni. Per ogni foglio, si verifica (attraverso espressioni regolari) che la regola non imponga z-index negativo o opacità nulla. Nel caso la regola fosse un @import, vengono analizzate le regole del foglio di stile importato.

```
function testStyle(doc) {  
  
    var negIndexFound=false;  
    var zeroOpacityFound=false;  
  
    sheets=doc.styleSheets;  
    for(i=0; i < sheets.length; i++){  
  
        rules=sheets[i].cssRules  
  
        for(j=0; j < rules.length; j++){  
  
            if(rules[j].cssText.search(/( )*@import/) > -1) {  
                importedRule=rules[j].styleSheet.cssRules;  
  
                for(k=0; k < importedRule.length; k++){  
                    if(importedRule[k].cssText.search(/z-index( )*/) > -1)  
                        negIndexFound=true;  
                    if(importedRule[k].cssText.search(/opacity( )*:( )*0( )*/) > -1)
```

```

        zeroOpacityFound=true;
    }

    }else{
        if(rules[j].cssText.search(/z-index( )*/) > -1)
            negIndexFound=true;
        if(rules[j].cssText.search(/opacity( )*:( )*0( )*/) > -1)
            zeroOpacityFound=true;
        }
    }
}
return (negIndexFound && zeroOpacityFound);
}

```

La presenza di iframe viene verificata generando un array contenente tutti i nodi dell'albero DOM del documento che abbiano tag pari a "iframe". Se l'array non è vuoto, significa che si hanno iframe nel documento.

```

function testIframes(doc){ //checks the presence of iframes

    var iframesUsed = false;
    var iframes = doc.getElementsByTagName("iframe");

    if(iframes.length != 0)
        iframesUsed=true;

    return iframesUsed;

}

```

Infine, le due funzioni seguenti, quando invocate, cambiano il valore dell'attributo checkresult in "alt" oppure "ok" e il tooltip visualizzato.

```

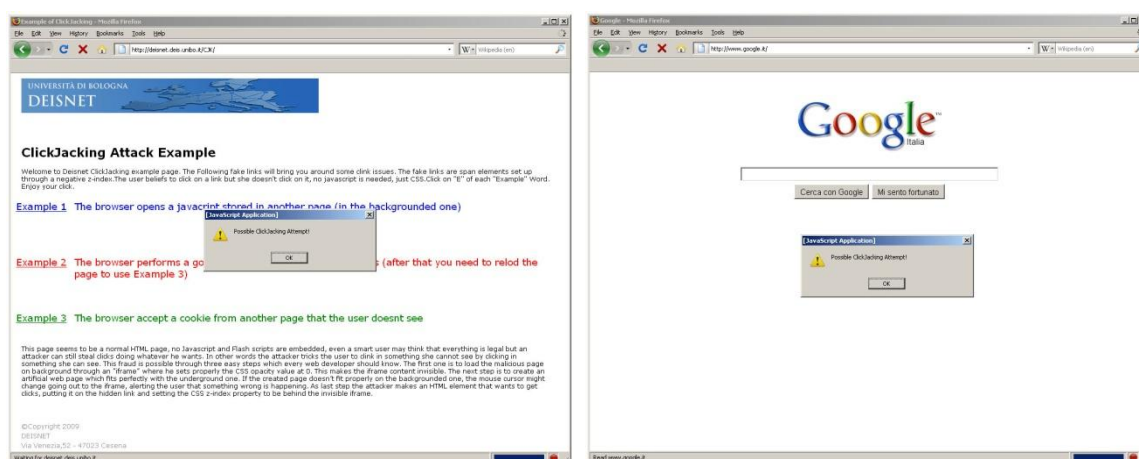
function alt(){
    var status = document.getElementById('clickjacking-statusbar-panel');
    status.setAttribute('checkresult', 'alt');
    status.setAttribute('tooltiptext', 'ClickJackThis! - Possible
ClickJacking Attempt!');
}

```

```
function ok() {  
    var status = document.getElementById('clickjacking-statusbar-panel');  
    status.setAttribute('checkresult', 'ok');  
    status.setAttribute('tooltiptext', 'ClickJackThis!');  
}
```

3.1.3 – Analisi del risultato

Il prodotto così ottenuto svolge il suo compito: una volta installato il file .xpi in Firefox, rileva correttamente le pagine riportate come esempi di clickjacking al primo capitolo. Tuttavia il numero di falsi positivi è molto alto e fra essi vi sono siti estremamente popolari e diffusi.



Si è ipotizzato che ciò sia dovuto al fatto che per avere siti graficamente accattivanti, la presenza di elementi invisibili risulta fondamentale, nonché la presenza di iframe. Un ulteriore punto debole rilevato è l'incapacità di gestire più livelli di importazione di CSS, ovvero il caso di un foglio di stile che importi un altro con @import il quale faccia lo stesso con un altro e così via. Infine, non si tiene conto di oggetti attivi tipo Flash o Java, potenzialmente pericolosi. È quindi necessario risolvere questi problemi. Ciò è stato effettuato nella seconda versione.

3.2 – ClickJackThis! versione 0.2

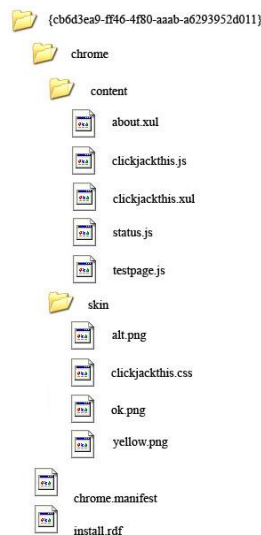
3.2.1 – L'idea

È stato necessario quindi ripensare ai criteri secondo i quali una pagina viene marcata come pericolosa. A tal fine, si è introdotto un ulteriore livello di allerta, intermedio, simboleggiato dal colore giallo, nel quale il software non è può affermare con certezza la

presenza di pericoli effettivi per la sicurezza ma tuttavia si rilevano possibili elementi di rischio. Allora, verranno ora classificate “rosse” le pagine che abbiano oggetti con z-index negativo e iframe invisibili, ovvero cui sia applicata una regola opacity:0. Saranno “gialle” quelle con oggetti attivi e che usino z-index negativi. Infine, saranno “verdi” le altre. Il problema di avere più @import annidati è stato risolto usando una funzione ricorsiva. L’aspetto negativo che subito si mostra è l’aumentata complessità del codice, che ha richiesto la suddivisione della parte in JavaScript in file diversi.

3.2.2 – L’implementazione

La struttura di questa seconda versione è la seguente:



Si è aggiornato il numero di versione nel file install.rdf e, mentre resta invariato il file manifest. Si è poi aggiunto un terzo selettore al file clickjackthis.css e ovviamente un file png 🟡 rappresentante un semaforo giallo:

```
#clickjacking-statusbar-panel[checkresult="yellow"] {
    list-style-image: url("chrome://ClickJackThis/skin/yellow.png");
}
```

Si è corretto il numero di versione nella finestra di dialogo. Al file clickjackthis.xul si sono aggiunte le righe per includere gli script esterni, ora divisi su tre file:

```
<script type="application/x-javascript"
    src="chrome://ClickJackThis/content/ClickJackThis.js" />
```

```
<script type="application/x-javascript"
```

```
src="chrome://ClickJackThis/content/Status.js" />

<script type="application/x-javascript"
src="chrome://ClickJackThis/content/TestPage.js" />
```

La modifica più rilevante è quindi stata effettuata sulla parte JavaScript. Si è creato un file principale, nominato `clickjackthis.js`, il cui compito è quello di associare e implementare gli ascoltatori per gli eventi associati alle pagine e alle schede. Tali ascoltatori usano le funzioni descritte nel file `testpage.js` per effettuare controlli sulle pagine e quelle in `status.js` per cambiare icona e tooltip visualizzato. L'aggiunta degli ascoltatori è effettuata in modo analogo:

```
window.addEventListener("load",function(){
gBrowser.addEventListener("load", OnPageLoad, true);},false);
window.addEventListener("pagehide", OnPageUnload, false);
```

Si è reso necessario quindi un secondo array per memorizzare le pagine del livello intermedio.

```
var redPages = new Array();
var yellowPages = new Array();
```

La funzione `OnPageLoad()` non subisce alcuna modifica formale, anche se la funzione `testpage()` cui fa riferimento è stata pesantemente cambiata.

```
function OnPageLoad(event){
  if (event.originalTarget instanceof HTMLDocument) {
    if (event.originalTarget.defaultView.frameElement)
      return;

    var doc = event.originalTarget;
    while (doc.defaultView.frameElement)
      doc=doc.defaultView.frameElement.ownerDocument;

    var container = gBrowser.tabContainer;
    var container2 = gBrowser.mPanelContainer;
    container.addEventListener("TabClose", OnPageUnload, false);
    container.addEventListener("TabOpen", tabOpened, false);
    container2.addEventListener("select", tabSelected, false);
```

```

        testPage (doc) ;
    }
}

```

Alla chiusura di una pagina o di una scheda, si deve verificare che il riferimento ad essa non sia presente né nell'array delle pagine maligne né in quello delle pagine in dubbio. Diversamente, si deve procedere alla rimozione dall'opportuno array del riferimento.

```
function OnPageUnload(aEvent) {
```

```
    if (aEvent.originalTarget instanceof HTMLDocument) {
```

Si scorrono gli array delle pagine e si confronta ogni elemento col riferimento dato dall'evento che chiama la funzione.

```

        var doc = aEvent.originalTarget
        for(var i=0; i<redPages.length; i++){

            if(redPages[i]==doc){
                redPages.splice(i,1);
                break;
            }
        }
        for(var i=0; i<yellowPages.length; i++){

            if(yellowPages[i]==doc){
                yellowPages.splice(i,1);
                break;
            }
        }
    }
}

```

Si deve poi gestire l'apertura in background di un link in una nuova scheda:

```
function tabOpened(event) {
    if(gBrowser.selectedBrowser != event.target.linkedBrowser)
testPage(event.target.linkedBrowser.contentDocument);
}

```

Infine, quando sono aperte più schede, si deve gestire il passaggio da una all'altra visualizzando per ognuna la corretta icona, possibilmente minimizzando gli scorrimenti di array, per non inficiare troppo le prestazioni del browser. Alla selezione di una scheda quindi, si cerca il riferimento della pagina in uno dei due array e, non appena viene trovato, si cambia l'icona e si arresta la ricerca:

```
function tabSelected(event){
    var doc =
gBrowser.getBrowserAtIndex(gBrowser.mTabContainer.selectedIndex).content
Document;

    var sec = true;

    for(var i=0; i<redPages.length; i++){

        if(redPages[i]==doc){
            alt();
            sec=false;
            break;
        }
    }

    for(var i=0; i<yellowPages.length; i++){

        if(yellowPages[i]==doc){
            warn();
            sec=false;
            break;
        }
    }

    if(sec)
        ok();
}
```

Le funzioni `alt()`, `ok()`, e `warn()` sono implementate separatamente nel file `status.js`:

```
function ok(){
    var status = document.getElementById('clickjacking-statusbar-panel');
    status.setAttribute('checkresult', 'green');
    status.setAttribute('tooltiptext', 'ClickJackThis!');
```

```

}

function warn(){
    var status = document.getElementById('clickjacking-statusbar-panel');
    status.setAttribute('checkresult', 'yellow');
    status.setAttribute('tooltiptext', 'ClickJackThis! - Possible
ClickJacking Attempt!');
}

function alt(){
    var status = document.getElementById('clickjacking-statusbar-panel');
    status.setAttribute('checkresult', 'red');
    status.setAttribute('tooltiptext', 'ClickJackThis! - ClickJacking
Attempt!');
}

```

L'analisi della pagina viene fatta, come visto, dalla funzione `testPage()`, contenuta in `testpage.js`. Essa prevede di cercare: regole CSS che impongano z-index negativi, presenza di oggetti attivi, presenza di iframes con attributo `opacity` nullo. La ricerca di regole CSS viene effettuata da una funzione apposita, descritta oltre, che restituisce un array di regole contenenti l'espressione regolare passata per argomento; se l'array è vuoto, non si hanno occorrenze della regola cercata.

```

function testPage(doc){

    var zIndexFound=false;
    var objectsFound=false;
    var invisibleIFrames=false;
    var a=findCss(doc, "z-index");

    if(a.length != 0)
        zIndexFound=true;
}

```

La ricerca di oggetti è demandata a una ulteriore funzione `findObjects()` che ritorna un risultato booleano; analogo per la ricerca di iframes.

```

objectsFound=findObjects(doc);
invisibleIFrames=findInvisibleIFrames(doc);

```

Si è ora in possesso degli indicatori (booleani) che si è deciso, in fase di progetto, di considerare per la classificazione della pagina. Quindi non resta che applicare il criterio

scelto e nel caso di pagina giudicata non sicura o dubbia, dare immediata comunicazione all'utente, salvarne il riferimento nell'array opportuno e cambiare l'icona.

```
if(zIndexFound && invisibleIFrames){
    alt();
    alert("ClickJacking Attempt!");
    redPages.push(doc);

}else if(zIndexFound && objectsFound){
    warn();
    alert("Possible ClickJacking Attempt!");
    yellowPages.push(doc);

}else{
    ok();
}
}
```

La ricerca nei CSS doveva prevedere livelli multipli di @import. Si è resa necessaria quindi l'implementazione di una funzione ricorsiva nav(sheet, regex, a), col compito di aggiungere all'array a ogni regola contenuta nel foglio di stile sheet che verifichi l'espressione regolare regex; per far ciò, essa scorre regola per regola il foglio; quando trova una direttiva @import, essa invoca se stessa sul foglio importato, passando lo stesso riferimento all'array a. In risultato finale è che alla fine della sua esecuzione, ogni CSS è stato valutato e l'array contiene tutte le regole cercate. Tale array viene restituito come risultato dalla funzione findCss().

```
function findCss(doc, regex){
    var a = new Array();
    var sheets = doc.styleSheets;

    for(var i=0; i<sheets.length; i++)
        nav(sheets[i], regex, a);

    return a;
}
```

```
function nav(sheet, regex, a){
    var rules = sheet.cssRules;
```

```

for(var i=0; i<rules.length; i++){
  if(rules[i].cssText.search("@import") > -1){
    nav(rules[i].styleSheet, regex, a);
  }else{
    if(rules[i].cssText.search(regex) > -1)
      a.push(rules[i].cssText)
  }
}
}

```

La ricerca di oggetti avviene navigando nodo per nodo l'albero DOM del documento; il metodo `getElementsByTagName` restituisce un array contenente tutti i nodi che corrispondono al nome passato per argomento. Per includere un oggetto, sia esso Flash o altro, HTML prevede di usare il tag `<embed>` e di specificare attraverso l'attributo `type` il tipo di oggetto che si sta includendo, attraverso il suo tipo MIME [5]. Quindi una volta in possesso dell'array di nodi di tipo `embed`, si cerca quali di essi corrisponde al MIME richiesto. Tale operazione può in certi casi lanciare un'eccezione (ad esempio per tag `embed` che non definiscono l'attributo `type`) che va gestita con un blocco `try/catch` onde evitare spiacevoli messaggi all'utilizzatore dell'estensione. Ragionamento analogo per la ricerca di Java Applet, le quali hanno un tag HTML dedicato, `<applet>`, che può essere ricercato per nome nell'albero DOM. Il risultato della funzione è un valore booleano che indica la presenza o meno di oggetti nel documento.

```

function findObjects(doc){

  var found=false;
  var objects = doc.getElementsByTagName("embed");
  var applets = doc.getElementsByTagName("applet");

  try{

    for(var i=0; i<objects.length; i++)
      if(objects[i].getAttribute("type" == "application/x-shockwave-flash")
        found=true;
  }catch(e){}
}

```

```
if(applets.length != 0)
    found=true;

return found;
}
```

La ricerca di nodi DOM per nome porta ad avere agevolmente un array di riferimenti agli iframes. JavaScript non implementa direttamente metodi per avere le regole di stile ad esso applicate. È stato quindi necessario, per ogni iframe, rilevare il suo id o la sua classe ed andare ad estrarre dai CSS, attraverso la già descritta funzione findCss(), le regole associate a quell'id o classe, sottoforma di array. Scorrendolo, si verifica se esistono direttive che impongano opacità nulla. Come sopra, il metodo getAttribute può lanciare eccezioni laddove non sia definito l'attributo richiesto; per questo si è usato un blocco try/catch.

```
function findInvisibleIFrames (doc) {

    var iframes = doc.getElementsByTagName("iframe");
    var found = false;

    for(var i=0; i<iframes.length; i++){
        try{

            var idRules = findCss(doc,
RegExp("#"+iframes[i].getAttribute("id")));
            var classRules = findCss(doc,
RegExp("\."+iframes[i].getAttribute("class")));

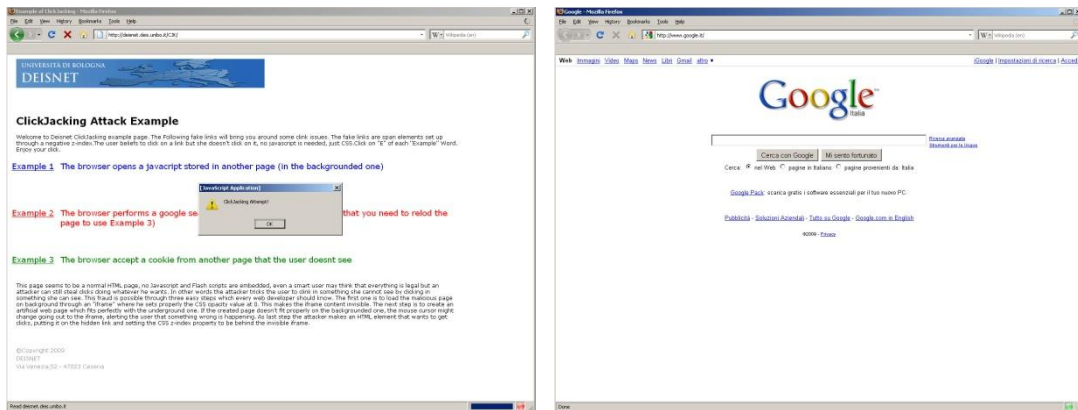
            for(var j=0; j<idRules.length; j++)
                if(idRules[j].search(/opacity( )*:( )*0( )*/;) > -1)
                    found=true;

            for(var j=0; j<classRules.length; j++)
                if(classRules[j].search(/opacity( )*:( )*0( )*/;) > -1)
                    found=true;

        }catch(e){}
    }
    return found;
}
```


3.2.3 – Analisi del risultato

Il prodotto così ottenuto risolve tutti le mancanze rilevate nella precedente versione. Rileva gli esempi di clickjacking già visti e non presenta nessun falso positivo per le pagine maligne, per lo meno nei test effettuati da chi scrive. Sono in numero considerevole invece le pagine “gialle”.



Questo porta ad una navigazione spesso interrotta da messaggi di allerta. Sarebbe molto più comodo per l'utente poter scegliere se essere avvisato tramite pop-up della rilevazione. Inoltre, a fini statistici, sarebbe utile poter disporre di uno storico riportante le pagine visitate e la loro valutazione da parte dell'estensione. Ovviamente tale funzione deve poter essere disattivabile, per evidenti motivi di riservatezza. La terza versione lavora su questi due aspetti.

3.3 – ClickJackThis! versione 0.3

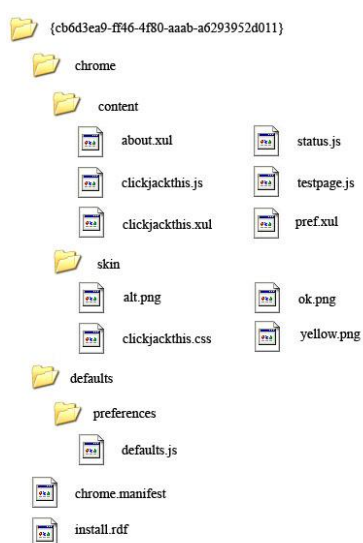
3.3.1 – L'idea

Si è ritenuto che la parte di analisi e valutazione delle pagine web non richiedesse ulteriori modifiche. Quindi, si è trattato il problema dell'aggiungere le preferenze all'estensione. Mozilla mette a disposizione un componente XPCOM che gestisce in modo semplice ed efficace le preferenze. Si è quindi creata una nuova finestra di dialogo per interfacciare l'utente a questo componente. Riguardo l'esigenza di avere statistiche sulle pagine visualizzate, era necessario memorizzare in qualche forma i dati. Come visto, esistono all'interno del Mozilla Application Framework strumenti per l'utilizzo di database ed altre forme di memorizzazione avanzata. Tuttavia, i tempi di scrittura di nuove voci è dettato dal tempo di navigazione da una pagina web all'altra dell'utente,

per cui è senza dubbio adatta allo scopo la memorizzazione in file di testo, che ha i pregi di richiedere una molto minore complessità a livello di codice e di creare direttamente dei file elaborabili da comuni software da ufficio. Ciò nonostante, il JavaScript standard, nasce come linguaggio di scripting per pagine web, per cui non prevede metodi per accedere al file. È stato pertanto necessario di nuovo appellarsi pesantemente ai componenti XPCOM.

3.3.2 – Implementazione

La nuova struttura dell'estensione prevede file aggiuntivi:



Lasciato invariato il file manifest, si è aggiornato il numero di versione nel file install.rdf. L'intero contenuto della cartella skin non è stato modificato. Le possibilità offerte all'utente sono tre: scegliere di essere allertato o meno al rilevamento di pagine ritenute maligne, al rilevamento di pagine dubbie e di abilitare o disabilitare la scrittura su file delle statistiche. Si è allora innanzi tutto progettata la finestra di dialogo delle preferenze, per permettere all'utente di governare il comportamento dell'estensione. Essa è descritta in pref.xul ed è qualificata come <prefwindow> in modo che venga riconosciuta automaticamente come quella da visualizzare alla pressione del tasto opzioni nel gestore delle estensioni [3-f].

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>

<prefwindow id="pref"
```

```
title="ClickJackThis! - Configuration Panel"
```

```
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
```

Quindi si è creato un pannello all'interno del quale si sono per prima cosa dichiarate nome e tipo delle preferenze che si intende gestire in esso; esse sono identificate da una stringa che deve essere unica in tutto l'ambiente Firefox. Mozilla pertanto raccomanda di seguire per il nome il modello "extensions.<extension_name>.<pref_name>". Quindi si è aggiunto al pannello tre checkbox, una per ogni preferenza, con opportuna descrizione testuale a lato.

```
<prefpane id="prefPane" label="Settings">
```

```
  <preferences>
```

```
    <preference id="alert" name="extensions.ClickJackThis.alert"
type="bool"/>
```

```
    <preference id="warn" name="extensions.ClickJackThis.warn"
type="bool"/>
```

```
    <preference id="log" name="extensions.ClickJackThis.log"
type="bool"/>
```

```
  </preferences>
```

```
  <vbox align="left">
```

```
    <checkbox preference="alert" label="Alert me of ClickJacking
attempts."/>
```

```
    <checkbox preference="warn" label="Warn me about possible
ClickJacking attempts."/>
```

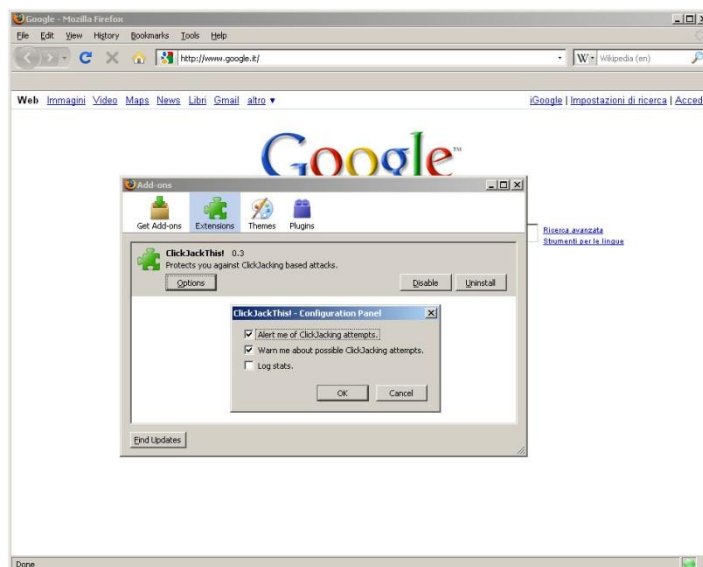
```
    <checkbox preference="log" label="Log stats."/>
```

```
  </vbox>
```

```
</prefpane>
```

```
</prefwindow>
```

Il risultato ottenuto è il seguente:



Si è quindi modificato la risposta al click destro sull'icona dell'estensione. Ora compare un piccolo menù contestuale con le voci about e configure. Esse rispettivamente mostrano la finestra di dialogo about.xul o la finestra delle preferenze pref.xul. Per far ciò si è aggiunto in coda al file clickjackthis.xul la seguente descrizione:

...

```
<popupset>
  <menupopup id="menu">
    <menuitem label="About"
              default="true"
              command="window.openDialog('chrome://ClickJackThis/content/about.xul',
'', 'centerscreen,chrome,resizable=no');"/>
    <menuseparator/>
    <menuitem label="Configure"
              oncommand="window.openDialog('chrome://ClickJackThis/content/pref.xul',
'', 'centerscreen,chrome,resizable=no');" />
  </menupopup>
</popupset>
</overlay>
```

Firefox richiede che le preferenze vengano obbligatoriamente inizializzate con un valore di default. È quindi richiesto il file defaults.js, nella cartella opportuna, il quale deve contenere solo ed esclusivamente le chiamate alla funzione pref(), imponendo ad ogni opzione, come già detto identificata dal suo nome, un valore di default.

```
pref("extensions.ClickJackThis.alert", true);
```

```
pref("extensions.ClickJackThis.warn", true);
pref("extensions.ClickJackThis.log", false);
```

Fatto ciò si è passati alla modifica delle parti già esistenti del progetto. I file `about.xul` e `clickjackthis.js` sono identici a quelli della versione precedente. Il file `testpage.js` ha subito alcune aggiunte. Ci si è innanzi tutto procurati un riferimento al componente XPCOM responsabile della gestione delle preferenze, chiamato `nsIPrefBranch`:

```
var prefManager = Components.classes["@mozilla.org/preferences-
service;1"].getService(Components.interfaces.nsIPrefBranch);
```

Quindi, effettuati come precedentemente i controlli sulla pagina, si procede a verificare attraverso tale componente, più precisamente invocandone il metodo `getBoolPref` e passando come argomento il nome scelto per la preferenza, se l'utente vuole essere allertato o meno e se ha abilitato le statistiche. La funzione `writeToLog`, descritta in dettaglio in seguito, ha il compito di scrivere su un opportuno file la stringa passata per argomento.

```
...
if(zIndexFound && invisibleIFrames){
    alt();
    if(prefManager.getBoolPref("extensions.ClickJackThis.alert"))
        alert("ClickJacking Attempt!");
    redPages.push(doc);
    if(prefManager.getBoolPref("extensions.ClickJackThis.log"))
        writeToLog(doc.URL + "\t red \n");
}
else if(zIndexFound && objectsFound){
    warn();
    if(prefManager.getBoolPref("extensions.ClickJackThis.warn"))
        alert("Possible ClickJacking Attempt!");
    yellowPages.push(doc);
    if(prefManager.getBoolPref("extensions.ClickJackThis.log"))
        writeToLog(doc.URL + "\t yellow \n");
}
else{
    ok();
    if(prefManager.getBoolPref("extensions.ClickJackThis.log"))
        writeToLog(doc.URL + "\t green \n");
}
}
...

```

Quindi lo script continua come precedentemente. La funzione `writeToLog()` crea o apre un file chiamato `ClickJackThis.log`. Per primo si è ottenuto un riferimento al componente XPCOM `nsIFile` e si fa in modo, come scelta di progetto, che punti alla cartella Desktop dell'utente; si specifica poi che il file di riferimento deve essere quello di nome "ClickJackThis.log".

```
function writeToLog(data) {
    var file =
Components.classes["@mozilla.org/file/directory_service;1"].
    getService(Components.interfaces.nsIProperties).
    get("Desk", Components.interfaces.nsIFile);
    file.append("ClickJackThis.log");
```

Quindi viene aperto un output stream, altro componente XPCOM; se il file puntato dalla variabile `file` non esiste, viene creato e inizializzato (per esempio impostandone i permessi d'accesso) dall'output stream; altrimenti viene aperto in scrittura a partire dal fondo.

```
    var foStream = Components.classes["@mozilla.org/network/file-output-
stream;1"].createInstance(Components.interfaces.nsIFileOutputStream);

    if(!file.exists()) {
        file.create(Components.interfaces.nsIFile.NORMAL_FILE_TYPE, 0666);
        foStream.init(file, 0x02 | 0x08 | 0x20, 0666, 0);
    }else{
        foStream.init(file, 0x02 | 0x10, 0666, 0);
    }
}
```

Per scrivere sul file delle stringhe invece che byte, serve un componente, detto `converter-output-stream`, che prende una stringa codificata in formato UTF-8 e si occupa di interagire con l'output stream in modo che venga effettuato quanto richiesto. Infine si chiude il file, per evitare errori di coerenza nel file system.

```
    var converter = Components.classes["@mozilla.org/intl/converter-
output-stream;1"].createInstance(Components.interfaces.nsIConverter
OutputStream);
    converter.init(foStream, "UTF-8", 0, 0);
    converter.writeString(data);
    converter.close();
}
```

3.3.3 – Analisi del risultato

L'estensione così modificata risponde a tutte le specifiche preposte. Coerentemente con la precedente versione, per quanto testato, solo le due pagine d'esempio vengono marcate come maligne. Sono poche quelle che risultano dubbie. In ogni caso, la possibilità offerta di disabilitare gli avvisi rende agevole la navigazione. Infine, l'elaborazione dei file di statistica da modo di trarre conclusioni utili per il miglioramento dell'estensione stessa.

Conclusioni

Si è visto come il clickjacking sia un tipo di attacco molto potente, sia per la sua agevole implementazione, sia per la sua efficacia e duttilità, sia per l'oggettiva difficoltà di porvi rimedio, dato che si basa su tecnologie comuni e oramai irrinunciabili nella moderna società dell'informazione. La soluzione proposta quindi si è orientata alla protezione del singolo utente, attraverso la progettazione e realizzazione di un'estensione dedicata di uno dei browser più diffusi a livello mondiale, Mozilla Firefox. Il suo sviluppo ha richiesto più fasi, dato che la risoluzione di un problema portava alla comparsa di altri. Il risultato finale è un prodotto che svolge il compito per cui è stato creato. Infatti, esso rileva le pagine che sicuramente implementano attacchi di tipo clickjacking e nessun falso positivo almeno nei test effettuati. Il numero di pagine giudicate di dubbia sicurezza è ancora alto; per arginare i fastidi introdotti dai continui avvisi, è possibile disabilitare i pop-up. Si riportano le statistiche raccolte su un totale approssimativo di 5 ore di navigazione su internet, raccolte da 5 utenti di diversa età e occupazione, per cui con diverse abitudini: 2 studenti di età compresa fra 20 e 25 anni, 1 studente di 15 anni, due adulti di età fra 50 e 60 anni.

Pagine SICURE	2929
Pagine DUBBIE	176
Pagine MALIGNI	2*

*: si sono volutamente visitati i due siti menzionati.

All'incirca 1 pagina ogni 17 è stata rilevata di dubbia sicurezza, mentre 1 ogni 1464 è stata rilevata maligna, anche se questo dato non è rilevante in quanto si sono volutamente visitati i due siti sopra menzionati. Si può quindi affermare che in circa tremila pagine non si è avuto alcun falso positivo. Lo strumento creato offre quindi una efficace protezione per gli attacchi clickjacking finora conosciuti. Come sempre accade nel campo della sicurezza informatica, ad un nuovo sistema di protezione corrisponde sempre l'avanzamento delle tecnologie di attacco. Possono essere pensate infatti nuove implementazioni con cui creare pagine malevole: ad esempio, si potrebbe porre in una pagina un iframe visibile, che contenga ad esempio due iframe sovrapposti, di cui uno invisibile. Allo stato attuale, tale pagina non verrebbe rilevata. Altre possibilità sono date

dall'utilizzo di JavaScript che permette, come visto, di avere controllo totale sulla pagina e sul comportamento del browser. In definitiva, è quindi necessario tenere aggiornato l'algoritmo di valutazione dell'estensione al fine di mantenere l'utilità dello strumento.

Bibliografia

- [1] F. Callegati & M. Ramilli, "Frightened by Links"
- [2] <http://extensions.roachfiend.com/cgi-bin/guid.pl>
- [3] <https://developer.mozilla.org/en>, in particolare:
 - a. Concetti Generali:
https://developer.mozilla.org/en/Building_an_Extension
 - b. Gestione eventi di caricamento pagina:
https://developer.mozilla.org/en/Code_snippets/On_page_load
 - c. Gestione navigazione per schede:
https://developer.mozilla.org/en/Code_snippets/Tabbed_browser
 - d. Mozilla Application Framework
https://developer.mozilla.org/en/mozilla_application_framework_in_detail
 - e. Pannello nella barra di stato:
<https://developer.mozilla.org/En/XUL/Statusbarpanel>
 - f. Preferenze:
https://developer.mozilla.org/en/Adding_preferences_to_an_extension
 - g. XUL:
<https://developer.mozilla.org/en/XUL>
- [4] <http://www.w3.org/> & <http://www.w3schools.com/>, in particolare:
 - a. CSS:
<http://www.w3.org/TR/CSS2/>
 - b. DOM:
<http://www.w3.org/DOM/>
 - c. DTD:
<http://www.w3schools.com/dtd/default.asp>
 - d. HTML:
<http://www.w3.org/html/>
 - e. Javascript:
<http://www.w3schools.com/jsref/default.asp>
 - f. RDF:
<http://www.w3.org/TR/REC-rdf-syntax/>

g. Selettori:

<http://www.w3.org/TR/CSS21/selector.html>

h. Statistiche sulla diffusione dei vari browser:

http://www.w3schools.com/browsers/browsers_stats.asp

i. XML:

<http://www.w3.org/XML/>

[5] <http://www.iana.org/assignments/media-types/>

[6] http://en.wikipedia.org/wiki/History_of_Mozilla_Application_Suite