Clickjacking-Angriff auf Webseiten

Gel Han
Betreuer: Dr. Heiko Niedermayer
Hauptseminar Innovative Internet-Technologien und Mobilkommunikation WS2010/2011
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: hang@in.tum.de

KURZFASSUNG

Das Internet ist heutzutage ein sehr wichtiges Medium zum Austausch von Daten. Mit immer mehr Zugang zu sensiblen Daten steigt gleichzeitig die Anzahl der Angriffsmöglichkeiten im Internet. *Clickjacking* ist eine dieser Möglichkeiten, um einen Nutzer des Internets anzugreifen. Der Angreifer täuscht eine Webseite vor, um Aktionen des Benutzers (wie zum Beispiel Maus-Klicks) abzufangen. *NoScript* oder *ClickIDS* stellen zwei Abwehrmöglichkeiten dar, um sich vor Clickjacking zu schützen. Es gibt jedoch Browser-spezifische Probleme, die weiterhin ungelöst sind. Clickjacking wird zurzeit noch erfolgreich erkannt und geblockt. Sicherheitsprogramme müssen dennoch weiterhin verbessert werden, da Clickjacking in Zukunft ein ernstes Thema bezüglich Sicherheit im Internet sein wird.

Schlüsselworte

Clickjacking, Transparenter `iframe`, Framebusting, NoScript, ClearClick, ClickIDS, `X-FRAME-OPTIONS` Header

1. EINLEITUNG

Das Internet hat sich zu einer Plattform entwickelt, auf der sehr viele persönliche und auch sensible Daten im Umlauf sind. Soziale Netzwerke wie zum Beispiel *Twitter* oder *Facebook* werden immer populärer und gleichzeitig gibt es immer mehr Angriffsziele für einen Hacker. Heutzutage kommen Computer nicht ohne vorinstallierte Antivirenprogramme aus. Weitere Angriffstypen wie Würmer, Trojaner oder eine gezielte Denial of Service-Attacke können einem unwissende Benutzer Schaden hinzufügen. Neben diesen Typen gibt es auch andere Angriffe wie zum Beispiel *Clickjacking* (kurz CJ). Alles was der Angreifer braucht, um sein Opfer anzulocken, ist eine Webseite im Internet. Der Angreifer kann mithilfe eines CJ-Angriffs das Opfer dazu bringen, durch Maus-Klicks einen Account zu löschen, eine Ware zu kaufen oder seine Webcam einzuschalten. Der Benutzer hat dabei keine Ahnung was im Hintergrund geschieht, da alles versteckt abläuft. Bei den sechs Sicherheitszielen kann Clickjacking die Integrität der Daten, die Vertraulichkeit, die Verfügbarkeit, die Verbindlichkeit, sowie die Privatheit verletzen.

Im nächsten Kapitel werden wir etwas genauer auf den Begriff "Clickjacking" und die Vorgehensweise des Angriffs eingehen. Anhand von Beispielen werden erfolgreiche CJ-Angriffe vorgestellt. Um Schwachstellen gegenüber CJ zu beseitigen, wurden Abwehrmethoden entwickelt, die im darauf folgenden Kapitel erklärt werden. Zuletzt folgt ein Überblick über verwandte Arbeiten sowie eine kurze Zusammenfassung und ein Ausblick auf die Zukunft von Clickjacking.

2. DER BEGRIFF "CLICKJACKING"

Der Begriff "Clickjacking" tauchte das erste Mal 2008 auf. "Clickjacking" wurde von Robert Hansen[1] und Jeremiah Grossman[2] geprägt. R. Hansen und J. Grossman stellten CJ-Angriffe und Schwachstellen auf der *OWASP NYC AppSec 2008* Konferenz vor (siehe [5]). Zum Entwickeln einer CJ-Attacke braucht der Angreifer die Markup-Sprachen *HTML* (HyperText Markup Language, siehe [6]) und *CSS* (Cascading Style Sheets, siehe [10]). Der Angreifer erstellt eine Webseite, um das unwissende Opfer anzulocken und auf Buttons oder Hyperlinks zu klicken. Dabei ist dem Opfer nicht bewusst, dass die Webseite eine Falle ist. Der Angreifer hat auf der Homepage eine zweite Webseite mittels `iframe` eingebunden und per CSS transparent gemacht. In Abbildung 1 sieht man ein Beispiel für eine Webseite mit einem eingebundenen `iframe` und einer potentiellen CJ-Attacke.
Clickjacking verwendet Elemente und Attribute, die in den Sprachen von HTML und CSS schon vorhanden sind. `iframes` werden mittels HTML in bestehende, normale Webseiten eingebunden. Um diese dann vor dem unwissenden Opfer zu verstecken, verwendet man CSS. Es gibt das Attribut `opacity`, um Objekte transparent zu machen. Der Benutzer denkt, dass er/sie sich auf einer ganz normalen Homepage befindet und auf Hyperlinks klickt. In Wirklichkeit liegt eine zweite Homepage über der angezeigten Webseite und die Klicks werden alle darauf registriert. In dem folgenden Listing sieht man den HTML-Quellcode, den der Angreifer braucht, um Facebook in einem transparenten `iframe` einzubinden:

```
<iframe src="http://www.facebook.com/home.php?"
     id="frame1" style="opacity:0.0;filter:
    alpha(opacity=0);" width="100%" height="
    100%"/></iframe>
```

[1]Robert Hansen ist CEO von der Sicherheitsconsulting-Firma "SecTheory". Er ist vor allem für seinen Beitrag zum Buch "XSS Attacks: Cross Site Scripting Exploits and Defense" und seiner Tätigkeit in der Hacker- und Sicherheitsszene bekannt.
[2]Jeremiah Grossman ist der Gründer und CTO von der Website-Security Firma "WhiteHat Security". Er gilt als einer der bekanntesten Experten im Bereich "Webapplication Security" und ist neben R. Hansen auch Co-Autor von "XSS Attacks: Cross Site Scripting Exploits and Defense".

doi: 10.2313/NET-2011-05-2_11

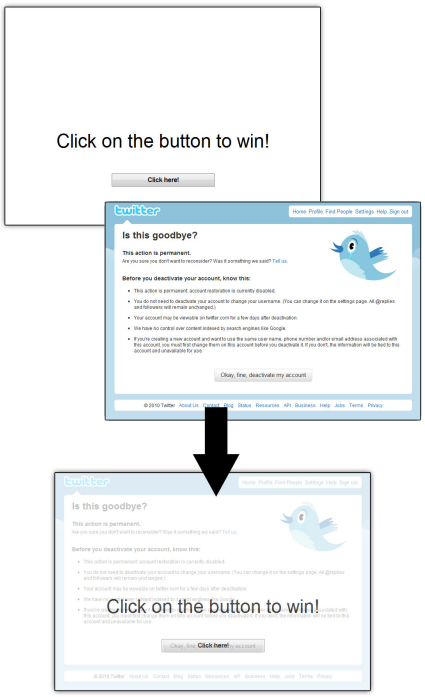

Abbildung 1: Der Angreifer baut sich den Clickjacking-Angriff aus zwei Webseiten zusammen: einer falschen, vom Angreifer erstellten HTML-Seite, die der unwissende Benutzer zu Gesicht bekommt. Die zweite Webseite wird in einem `iframe` eingebunden und als Angriffsziel genutzt. Im dritten Bild sieht man, dass der `iframe` mittels CSS transparent gemacht wurde.

Der CJ-Angriff wurde unter anderem durch Vorfälle auf dem Mikroblogging-Dienst *Twitter* (siehe [13]) und der Social Networking-Webseite *Facebook* bekannt. Im Fall von Twitter haben Angreifer eine Webseite mit einem Button erstellt. Bei einem Klick auf den Button wurde der Twitter-Status des Opfers aktualisiert. Diese CJ-Attacke funktioniert jedoch nur, wenn der Benutzer bei Twitter registriert und gerade angemeldet ist. Die neue Status-Nachricht liest sich wie folgt: "Don't Click: `http://xyz`". Wenn einer der "Followers" (Personen, die Twitter-Nachrichten des Opfers lesen) auf diese URL klickt, so gelangt derjenige auf die Webseite des Angreifers. Die Webseite des Angreifers hatte in einem transparenten `iframe` die Twitter-Homepage mit dem angemeldeten Benutzer geladen und in der Nachrichten-Box die Nachricht "Don't Click: `http://xyz`" kopiert sowie den "Don't Click"-Button auf den "Nachricht absenden"-Button gelegt. Mitarbeiter von Twitter haben auf dieses Problem reagiert (siehe [15]) und die Schwachstelle mittels JavaScript behoben. Diese Abwehrmethode (mit JavaScript) nennt sich *Framebusting* und wird in Kapitel 3.1 genauer erläutert.

Ein anderes Beispiel, wie ein Angreifer Gebrauch von einer Webseite mit transparentem `iframe` machen könnte, wäre mithilfe von Werbebannern. Der Angreifer erstellt eine Homepage mit einer CJ-Attacke, die das unwissende Opfer mehrmals auf einen Button klicken lässt. Jeder dieser Maus-Klicks geschieht auf dem Werbebanner und bringt somit dem Angreifer Geld. Die Webseite könnte ein Spiel vortäuschen, bei dem der Benutzer so schnell und so oft wie möglich auf einen Button klicken soll um in eine Highscore-Liste eingetragen zu werden oder etwas zu gewinnen.

Auf der OWASP NYC AppSec 2008 Konferenz haben R. Hansen und J. Grossman eine Schwachstelle im Adobe Flash Player angesprochen, die durch Clickjacking ausgenutzt werden konnte, um eine installierte Webcam einzuschalten[3]. Mittlerweile hat Adobe diese Schwachstelle behoben. In der Demo wurde eine Webseite mit einem Spiel erstellt, bei dem der Benutzer auf verschiedene Buttons klicken musste. Bei jedem Klick auf einem der Buttons verschwand dieser und ein neuer tauchte an einer anderen Stelle auf. Bei den meisten Klicks geschah nichts; bei vier Klicks wurden jedoch im Hintergrund Einstellungen des Flash Players verändert (siehe Abbildung 2). Der unwissende Benutzer hat auf diese Weise seine Webcam eingeschalten.

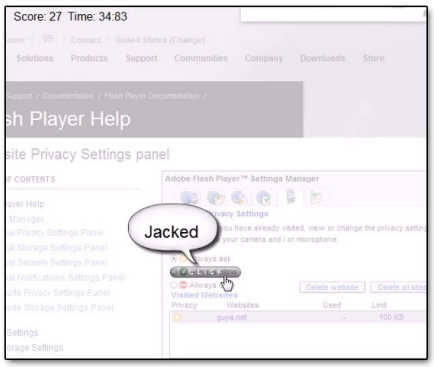

Abbildung 2: Die Abbildung zeigt einen Ausschnitt aus einem Clickjacking-Angriff auf den Adobe Flash Player. Ziel des Angreifer ist es, die Webcam des Benutzers einzuschalten, ohne dass dieser es bemerkt. Quelle: [1].

3. ABWEHRMETHODEN

Grundsätzlich gibt es zwei Möglichkeiten für Abwehrmaßnahmen gegen Clickjacking: 1. Der Benutzer schützt sich mit einem Programm oder 2. die Webseite wehrt Angriffe automatisch mittels Mechanismen ab. Eine Abwehrtechnik, die sehr weit verbreitet ist und in vielen HTML-Seiten angewendet wird, ist *Framebusting*. Diese Technik schützt eine Webseite davor, in einen `iframe` eingebettet zu werden. Framebusting basiert auf JavaScript und ist sehr einfach einzusetzen. Meist genügen ein paar Zeilen Code, um eine Webseite erfolgreich zu schützen. Dennoch gibt es Schwachstellen und Möglichkeiten, Framebusting zu umgehen. Weitere Details werden im Kapitel 3.1 erläutert. In Abschnitt 3.4 gehen wir auf den `X-FRAME-OPTIONS`-Header ein, der auch zur Abwehr von Angriffen wie Clickjacking gedacht ist. Im Anschluss wird das Feature *ClearClick* vom Plugin *NoScript* vorgestellt. NoScript wird hauptsächlich im beliebten Internetbrowser *Firefox* eingesetzt.

[3]Eine Demo des Clickjacking-Angriffs findet man auf `http://guya.net/security/clickjacking/game.html` [09.12.2010]

doi: 10.2313/NET-2011-05-2_11

3.1 Framebusting und Schwachstellen

3.1.1 Framebusting im Allgemeinen

Wie in der Einleitung schon erwähnt wurde, ist *Framebusting* eine Technik, die in Webseiten eingesetzt wird um zu verhindern, dass sie in andere Homepages eingebunden werden. Viele Webseiten, darunter Twitter, Facebook und Yahoo verwenden Framebusting, um sich gegen Clickjacking-Angriffe zu schützen. Der Framebusting-Code von Twitter macht folgendes (siehe Abbildung 1): es wird überprüft, ob das Fenster der Webseite auch das aktuelle Browser-Fenster ganz ausfüllt beziehungsweise das oberste Fenster im Frameset ist. Falls dies nicht der Fall sein sollte, so wurde die Webseite mindestens einmal in einen Frame eingebunden. Der nachfolgende Code ermöglicht es der Webseite, aus einem Frame "auszubrechen" und verhindert somit einen CJ-Angriff (Quelle: `http://www.twitter.com` [09.12.2010]):

```
if (window.top !== window.self) {document.write
    = "";window.top.location = window.self.
    location; setTimeout(function(){document.
    body.innerHTML='';},1);window.self.onload=
    function(evt){document.body.innerHTML
    ='';};}
```

Rydstedt et al. [12] haben im Juli 2010 eine Untersuchung zum Thema "Framebusting" durchgeführt. Es wurden die Top-500 Seiten vom Dienst "Alexa" auf JavaScript-Code untersucht. Dabei wurde festgestellt, dass die meisten Seiten als Abfrage `if (top != self)` und als Statement darauf `top.location = self.location` verwenden. Wie an diesem Beispiel zu sehen ist, besteht ein Framebusting-Code aus zwei Teilen: einem Abfrage-Statement und einer Anweisung, falls ersteres zutrifft. Die Untersuchung lieferte folgendes Ergebnis (für die vollständige Tabelle siehe [12]):

Am meisten verwendete Abfragen für Framebusting:

- `if (top != self)`
- `if (top.location != self.location)`
- `if (top.location != location)`

Am meisten verwendete Statements:

- `top.location = self.location`
- `top.location.href = document.location.href`
- `top.location.href = self.location.href`
- `top.location.replace(self.location)`

`location` ist ein Objekt in JavaScript, das den Standort eines HTML-Dokuments beschreibt. Mit dem Attribut `href` kann man auf die komplette URL zugreifen. In den Statements kommen außerdem die Eigenschaften `top`, `parent` und `self` vor. Diese drei Eigenschaften werden bei einem Verweis auf das jeweilige Browserfenster verwendet. `top` stellt dabei das oberste, `self` das aktuelle und `parent` das übergeordnete Fenster in der Struktur dar. In JavaScript kann man sich mit diesen Eigenschaften durch die verschiedenen Fenster bewegen (jeweils eine Ebene nach oben oder nach unten). Für mehr Informationen zu JavaScript, siehe [3].

Bei der Untersuchung hat sich herausgestellt, dass die meisten Webseiten keinen Framebusting-Code auf ihren Startseiten hatten. Die Abwehrtechnik wurde meist nur bei den "Login"- oder den "Passwort zurücksetzen"-Seiten eingesetzt. In der Ausarbeitung von Rydstedt et al. wird außerdem darauf hingewiesen, dass die meisten Webseiten zwar ihre normale Homepage vor Clickjacking-Angriffen schützen, jedoch aber die mobilen Seiten vernachlässigen. Viele Webseiten bieten dem Benutzer die Möglichkeit, falls dieser über ein mobiles Endgerät verfügen sollte, auf eine mobile Version der Homepage zurückzugreifen. Der Benutzer gelangt jedoch auch mit einem normalen Browser auf die mobilen Seiten. Diese Webseiten bieten meist keinen Schutz vor Clickjacking-Angriffen und können so ein leichtes Ziel von Angreifern werden.

3.1.2 Schwachstellen

In diesem Unterkapitel werden einige Schwachstellen der Framebusting-Technik vorgestellt. Obwohl Framebusting sehr weit verbreitet ist und von Webseiten als *die* Lösung im Bezug auf Clickjacking angesehen wird, ist es sehr einfach, die Abwehrmethoden zu umgehen.

Ein Frame in einem Frame

Nach Rydstedt et al. gibt es Webseiten, die im Framebusting-Code auf Hierarchie prüfen. Dazu wird das Attribut `parent` aus JavaScript verwendet. Eine einfache Methode, diese Technik zu umgehen, ist ein Frame in einem Frame. Ein Beispiel für einen Framebusting-Code, der einen CJ-Angriff verhindern würde:

```
if(top.location != location) {
        parent.location = self.location }
```

Die Webseite würde aus einem `iframe` "ausbrechen" und der Benutzer würde die echte Seite zu Gesicht bekommen (auch bei zwei `iframes`). Falls jedoch dem Attribut `parent.location` etwas anderes zugewiesen wurde und die übergeordnete Webseite schon ein eingebetteter Frame war, so führt der Angreifer den unwissenden Benutzer auf eine zweite, falsche Webseite, auf der der Clickjacking-Angriff stattfindet. Dieser Sachverhalt ist in Abbildung 3 dargestellt.

`onBeforeUnload` und *204 Flushing*

Framebusting-Code führt dazu, dass eine Webseite, die eingebettet wurde, ausbricht und diese anstatt der falschen Webseite des Angreifers angezeigt wird. Diese Weiterleitung kann jedoch vom Angreifer manipuliert werden. Der JavaScript-Handler `onBeforeUnload` wird geladen, wenn der Browser auf die eingebette Seite wechseln will. Wenn der Angreifer dem Benutzer eine Nachricht mit einer Warnung anzeigt und dieser "Abbrechen" klickt, so bleibt das Opfer auf der Webseite mit dem eingebetteten Frame.

doi: 10.2313/NET-2011-05-2_11

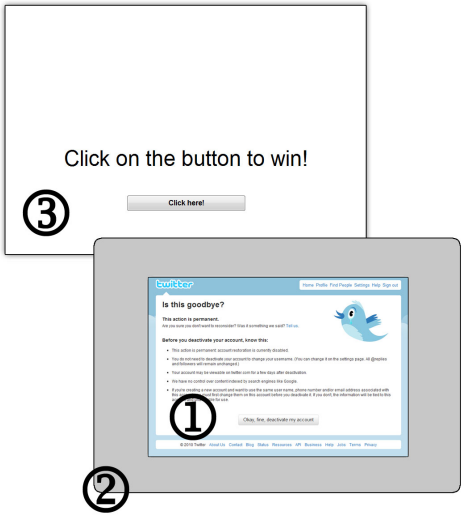

Abbildung 3: Sei (1) die authentische Seite mit Framebusting-Code. Diese erkennt, dass sie in einem Frame eingebunden wurde (2) und "bricht" aus. Die echte Seite (1) wird jedoch in einer weiteren Seite (3) eingebunden und kann somit für einen möglichen Angriff ausgenutzt werden.

Der folgende JavaScript-Code verhindert eine Weiterleitung, falls der Benutzer "Abbrechen" klickt:

```
window.onbeforeunload = function() {
        return ""; }
```

Eine andere Möglichkeit wäre, die Weiterleitung auf eine Webseite zu lenken, die einen *204 No Content*-Fehler zurückliefert. Während `onBeforeUnload` die Interaktion des unwissenden Benutzers erfordert hat, um auf der Webseite des Angreifers zu bleiben, so geschieht mit dieser Methode das "Abschalten" des Framebusting-Codes automatisch. Die Weiterleitung auf die *204 No Content*-Webseite führt dazu, dass alle registrierten Befehle gestrichen werden (daher auch der Name "Flushing"). Somit wird auch die Weiterleitung auf die echte Webseite durch Framebusting ignoriert.

XSS-Filter

Mit XSS-Filtern, die in den Browsern *Google Chrome* und *Microsoft Internet Explorer 8* integriert sind, ist es möglich, Framebusting-Code zu umgehen. Die *reflective XSS Filter* dienen zur Abwehr von XSS-Angriffen. Der Internet Explorer untersucht dabei Abfragen nach möglichen Cross-Site-Scripting-Attacken und falls Übereinstimmungen gefunden werden, so werden diese Abschnitte übersprungen. Dies kann der Angreifer ausnutzen, in dem er im `iframe` einen Teil von einem Framebusting-Code als Übergabe-Parameter angibt. Der XSS-Filter wird beim Überprüfen feststellen, dass sich der Ausschnitt aus dem Code und der JavaScript-Code im Header (der Framebusting-Code zur Abwehr) übereinstimmen. Im Endeffekt wird der übereinstimmende Teil übersprungen. Für den Angreifer bedeutet dies, dass der `iframe` erfolgreich eingebunden werden kann. Während beim Internet Explorer alle *inline*-Scripts übersprungen werden (Teile des JavaScript-Codes und Cookies können trotzdem geladen werden), so kann man bei Chrome explizit Framebusting-Code ausschalten. Ein einfaches Beispiel (aus [12]) würde wie folgt aussehen:

Framebusting-Code:

```
if(top != self) {
        top.location=self.location; }
```

Angreifer:

```
<iframe src="http://www.victim.com/?v=if(top
    +!%3D+self)+%7B+top.location%3Dself.
    location%3B+%7D">
```

Hinter der URL im `iframe` hat der Angreifer den gesamten Framebusting-Code als Übergabeparameter definiert. Chrome stellt eine Übereinstimmung im Code fest und schaltet den JavaScript-Block aus.

Einbetten in Frames

Es gibt Webseiten, die erlauben, dass sie von einer ihrer eigenen Seiten eingebettet werden. Rydstedt et al. haben gezeigt, dass diese Methode bei den meisten Homepages Schwachstellen für einen CJ-Angriff aufweist. Zum Überprüfen, ob eine Webseite von einer bekannten Homepage eingebettet wird, wird der `document.referrer` verwendet. Dieser überprüft, ob die URL, von der die Anfrage zum Einbetten kommt auch mit der aktuellen URL übereinstimmt. In den meisten Fällen wird aber nicht die ganze URL abgeglichen. Es reicht, wenn der Angreifer zum Beispiel eine CJ-Attacke von einer Webseite mit der URL `http://www.walmart.com.badgy.com` startet (siehe [12]). Mit dieser Homepage könnte der Angreifer theoretisch die "Walmart"-Webseite als `iframe` einbinden, da diese zwar auf die URL `walmart.com` überprüft, jedoch nicht den Rest der URL.

Angenommen, eine Webseite wäre gegen Clickjacking-Angriffe geschützt und würde nur vertrauenswürdigen Seiten erlauben, sich selbst einzubetten. Eine einfache Schwachstelle, die ein Angreifer in dem Fall ausnutzen könnte, wäre das Verwenden des Dienstes *Google Images*. Der Angreifer sucht nach einem bestimmten Bild bei Google Images. Die gefundene Seite wird in einem Subframe angezeigt. Die meisten Webseiten vertrauen Google Images und lassen das Einbetten in den Frame zu (siehe Abbildung 4). Laut Rydstedt et al. [12] ist in diesem Fall das Problem Google selbst, da keine Abwehrtechniken gegen Clickjacking bei der Bildersuche verwendet werden. Der Angreifer kann in einen `iframe` Google Images, mit einer Anfrage auf ein Bild (das sich auf der Homepage des Angreifers befindet), einbinden. Im Beispiel von Rydstedt et al. konnte ein CJ-Angriff mittels Google Images und Social-Networking-Seite MySpace durchgeführt werden. Zuerst hat man nach einem Bild von dem MySpace-Profil gesucht und dann beide Webseiten in einen `iframe` eingebettet.

doi: 10.2313/NET-2011-05-2_11

Abbildung 4: Der Angreifer kann Google Images mittels `iframe` in seine Webseite einbinden. Das Ergebnis der Suche wird dabei auch mit eingebunden.

Das Attribut `restricted` beim Internet Explorer
Webseiten, die im *Restricted Zone*-Modus vom Internet Explorer aufgerufen werden, werden in ihrer Funktion eingeschränkt, da aus Sicherheitsgründen kein JavaScript und keine Cookies erlaubt sind. Angreifer können dies ausnutzen und einen `iframe` mit dem Attribut `security=restricted` verwenden. Falls die aufgerufene Seite einen Framebusting-Code enthalten sollte, so wird dieser durch den Internet Explorer ausgeschaltet. Die eingebettete Seite wäre somit ungeschützt vor Clickjacking-Angriffen und könnte überall eingebunden werden.

3.2 ClickIDS

Balduzzi et al. [8] haben in ihrem Paper *A Solution for the Automated Detection of Clickjacking Attacks* ein System vorgestellt, welches automatisch Clickjacking-Angriffe erkennt. Dieses System wurde *ClickIDS* genannt (IDS für *Intrusion Detection System*). ClickIDS überprüft alle anklickbaren Objekte einer Webseite anhand ihrer Koordinaten. Wenn sich zwei oder mehr Objekte bei einem Mausklick überlappen, wird ein auffälliges Verhalten registriert. Das System besteht aus zwei Teilen: einer Testumgebung und einer *Detection Unit*. Das automatisierte System spielt in der Testumgebung Szenarien durch und versucht, alle Objekte anzuklicken. Die Detection Unit analysiert eine Webseite und überprüft sie auf mögliche CJ-Angriffe. Sie besteht aus zwei Browser-Plugins wobei eines davon *NoScript* ist. Es wurden umfangreiche Tests mit dem System durchgeführt (es wurden zirka 70,000 URLs untersucht). Die Resultate haben gezeigt, dass ClickIDS zuverlässig Clickjacking-Angriffe erkennt. Es gibt zwar Grenzfälle und das System schlug oft falschen Alarm, dennoch wurden zwei Clickjacking-Attacken richtig durch das System erkannt.

3.3 NoScript/ClearClick

NoScript ist eine kostenlos erhältliche Erweiterung für den populären Internetbrowser *Firefox*. Die Erweiterung schützt vor Cross-Site-Scripting Angriffen und bietet ein Modul an, das auch Clickjacking-Angriffe erkennt und abwehren kann. Dieses Modul nennt sich *ClearClick*. Dieses Plugin erkennt Maus-Klicks auf transparenten Elementen und bei einem potentiellen Angriff bekommt der Benutzer eine Warnung (siehe Abbildung 5). Die aktuelle Aktion wird vorzeitig gestoppt. ClearClick verursacht sehr viele falsch-positive Resultate laut Balduzzi et al.

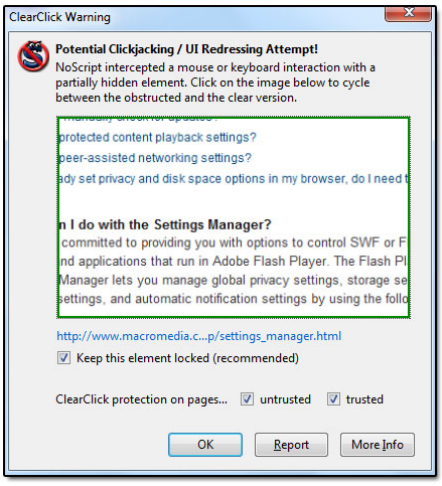

Abbildung 5: ClearClick erkennt potentielle Clickjacking-Angriffe und warnt den Benutzer im Voraus.

3.4 X-FRAME-OPTIONS Header

Das Entwicklerteam rund um Eric Lawrence von Microsoft hat im Internet Explorer 8 einen Mechanismus eingebaut, um Webseiten vor Clickjacking-Angriffen zu schützen (siehe [7]). Beim Erstellen von Webseiten können Entwickler einen `X-FRAME-OPTIONS`-Header angeben. Dieser stellt sicher, dass die Webseite in keiner anderen eingebettet wird und somit für einen Angriff ausgenutzt werden kann. Der Header erlaubt zwei Attribute: `DENY` und `SAMEORIGIN`. Während das erste Attribut allen Seiten verwehrt, die Webseite einzubetten, erlaubt das zweite Attribut der aktuellen Webseite beziehungsweise der gleichen Seite das Einbetten. Laut [12] verwenden nur vier Seiten aus den Top 10,000 der Alexa-Liste den `X-FRAME-OPTIONS`-Header. Dies kann auf die schwierige Einbindung des Headers zurückgeführt werden. Der Header kann in alle Seiten manuell eingebunden werden aber es treten Probleme auf, wenn ein Entwickler mehrere Domänen benutzen möchte, da es keine Liste mit individuell erlaubten Seiten gibt. Eine wesentlich elegantere Lösung wäre das serverseitige Mitsenden des `X-FRAME-OPTIONS` -Headers. Damit wären alle Webseiten mit dem Header ausgestattet.

`X-FRAME-OPTIONS` wird von den Browsern Microsoft Internet Explorer 8+, Apple Safari 4+ und Google Chrome 2+ unterstützt. Firefox unterstützt das Feature ab Version 3.6.9+.

4. VERWANDTE ARBEITEN

R. Hansen und J. Grossman stellten *Clickjacking* auf der OWASP NYC AppSec 2008 Konferenz vor (siehe [5]). Bei der Untersuchung der Angriffstechnik wurde eine Schwachstelle in Adobe's Flash Player gefunden. Adobe hat diese Lücke im Flash Player 10 behoben und das Security Team hat den beiden Experten für die Entdeckung gedankt (siehe [2]).

doi: 10.2313/NET-2011-05-2_11

Es existieren ähnliche Angriffe wie Clickjacking - eine der Angriffe, welche auch von Grossman auf der OWASP Konferenz erwähnt wurde, nennt sich *Cross-Site Request Forgery* (kurz CSRF). Diese Technik nutzt Schwachstellen in HTML aus und bringt den Browser des Opfers dazu, eine Aktion des Angreifers auszuführen. Dazu lockt der Angreifer den unwissenden Benutzer zuerst auf eine aufgesetzte, falsche Webseite. Diese sendet dann eine falsche Abfrage an die echte Webseite, die daraufhin die bösartige Aktion ausführt. Zeller et al. [16] haben in ihrem Paper nachgewiesen, dass es möglich ist, einen CSRF-Angriff auf die Webseite der Zeitung "New York Times" durchzuführen. Mittlerweile wurden die Schwachstellen seitens der New York Times behoben. CSRF-Angriffe sind sehr einfach zu entwickeln aber gleichzeitig auch sehr einfach zu beheben.

J. Grossman, einer der Entdecker von Clickjacking, gab ein ausführliches Interview in dem Artikel *Silver Bullet Talks with Jeremiah Grossman* [9], wo er unter anderem auf den Fall "Adobe" und Gefahren im Netz generell einging. Es wurde CSRF (und auch Cross-Site-Scripting) angesprochen und klar gemacht, dass Clickjacking unter anderem ein Problem ist, auf das Browser-Hersteller reagieren müssen.

Den ersten Report eines möglichen Clickjacking-Falls gab es in 2002. Jesse Ruderman hatte einen Bug-Report für *Mozilla* (siehe [11]) mit dem Titel *iframe content background defaults to transparent* erstellt. In einer kurzen Beschreibung erklärte er einen potentiellen Clickjacking-Angriff auf Yahoo. Sieben Jahre später veröffentlichte Grossman einen Blog-Eintrag (siehe [4]), in dem er voraussagte, dass Clickjacking-Angriffe vermutlich zwischen 2014 und 2017 immer mehr in den Vordergrund rücken werden.

Im White Paper von der Security-Firma *context* (siehe [14]) wird ein *Next Generation Clickjacking* beschrieben. Es handelt sich um einen Clickjacking-Angriff, bei dem der Benutzer eine Drag-and-Drop-Aktion ausführt und dabei zum Beispiel den gesamten Text von einer Facebook-Wall markiert und mit einem Klick auf einen Button an den Angreifer sendet. Das White Paper zeigt, dass es noch sehr viele Schwachstellen gibt und mit Framebusting alleine nicht alle Angriffe abgewehrt werden können.

5. ZUSAMMENFASSUNG

Framebusting sowie das System *ClickIDS* und die Erweiterung *NoScript* mit dem Plugin *ClearClick* haben gute Erfolge bei der Erkennung und Abwehr von *Clickjacking*-Angriffen gezeigt. Während es bei Framebusting einige Schwachstellen gibt, hat ClickIDS gute Ergebnisse geliefert. Es gibt zwar sehr viele falsch-positive Resultate, diese sind jedoch auf Grenzfälle zurückzuführen. Der `X-FRAME-OPTIONS` -Header ist eine Alternative zu den bestehenden Techniken.

Clickjacking-Angriffe waren für eine kurze Zeit populär, vor allem durch Attacken auf die Social Networking-Plattformen Twitter, Facebook und der Schwachstelle im Adobe Flash Player. J. Grossman [4] sagt, dass Clickjacking in den nächsten Jahren kein bedeutendes Sicherheitsrisiko spielen wird. Grossman erwähnt, dass Cross-Site-Scripting erst nach 8 Jahren (in 2005) ein großes Sicherheitsrisiko wurde. SQL Injections wurden erst nach 9 Jahren Entwicklung zu einem größeren Problem. Die erste Schwachstelle bezüglich Clickjacking wurde 2002 gefunden und 2008 wurde der Begriff "Clickjacking" das erste Mal erwähnt. Wenn man diese Entwicklungszeiten in Betracht zieht, so stellt Clickjacking vorerst keine Gefahr dar. Hersteller und Entwickler reagieren sehr schnell auf Schwachstellen und sorgen dafür, dass diese Sicherheitslücken schnell geschlossen werden. Die Resultate der Untersuchung von Balduzzi et al. [8] zeigen, dass es zurzeit noch nicht sehr viele Webseiten mit CJ-Angriffen gibt. Dennoch ist Vorsicht geboten und es ist ratsam, NoScript zu installieren falls man Firefox verwendet. Eine sehr einfache Lösung gegen Clickjacking wäre, dass User sich aus allen Webapplikationen sowie Webseiten nach einer Session richtig ausloggen. Die meisten Angriffe auf Seiten wie Twitter oder Facebook funktionierten, weil die Benutzer noch in den jeweiligen Seiten eingeloggt waren. Clickjacking-Angriffe nehmen von der Tatsache Gebrauch, dass Nutzer in Diensten wie Facebook oder Twitter meist eingeloggt bleiben. Sehr viele Nutzer schließen nur das Browser-Fenster. Dabei bleibt eine Session jedoch aktiv. Eine einfache Lösung wäre, dass der Nutzer nach einer bestimmten Zeit, in der er inaktiv war, automatisch ausgeloggt wird. Für Formulare gäbe es die Möglichkeit der Authentifizierung. Ein Benutzer muss etwas in ein Feld schreiben um das Formular wegschicken zu können, sei es ein Code oder eine Nummernfolge. Dies kann verhindern, dass CJ-Angriffe ausgenutzt werden, um vorgefüllte Formulare ohne Bestätigung des Nutzers abzusenden. Clickjacking-Angriffe sind sehr schnell zu entwickeln und der unerfahrene Benutzer kann schnell Opfer eines Angriffs werden, ohne dass der/diejenige etwas davon bemerkt.

6. LITERATUR

[1] Webcam ClickJacking. `http://www.youtube.com/watch?v=gxyLbpldmuU` [09.12.2010], Oktober 2008.

[2] Adobe (PSIRT). Thanks to Jeremiah Grossman and Robert "RSnake" Hansen. `http://blogs.adobe.com/psirt/2008/09/thanks_to_jeremiah_grossman_an.html` [09.12.2010], September 2008.

[3] D. Goodman. *JavaScript Bible*. Hungry Minds, Inc., 2001.

[4] J. Grossman. Clickjacking 2017. `http://jeremiahgrossman.blogspot.com/2009/06/clickjacking-2017.html` [09.12.2010], Juni 2009.

[5] J. Grossman and R. Hansen. New Zero-Day Browser Exploits - ClickJacking. `http://video.google.com/videoplay?docid=-5747622209791380934&hl=en#` [09.12.2010], 2008.

[6] H. Herold. *Das HTML/XHTML Buch: mit Cascading Style Sheets und einer Einführung in XML*. SuSe-PRESS, 2002.

[7] E. Lawrence. IE8 Security Part VII: ClickJacking Defenses. `http://blogs.msdn.com/b/ie/archive/2009/01/27/ie8-security-part-vii-clickjacking-defenses.aspx` [09.12.2010], Januar 2009.

[8] M.Balduzzi, M.Egele, E.Kirda, D.Balzarotti, and C.Kruegel. A Solution for the Automated Detection of Clickjacking Attacks. In *Proceedings of the ACM Symposium on Information, Computer and Communications Security (AsiaCCS)*, Beijing, China,

doi: 10.2313/NET-2011-05-2_11

April 2010.

[9] G. McGraw. Silver Bullet Talks with Jeremiah Grossman. *Security Privacy, IEEE*, 7(2):10 –14, 2009.

[10] W. Nefzger. *CSS: Cascading Style Sheets für Profis*. Franzis Verlag GmbH, 2006.

[11] J. Ruderman. iframe content background defaults to transparent. `https://bugzilla.mozilla.org/show_bug.cgi?id=154957` [09.12.2010], Juni 2002.

[12] G. Rydstedt, E. Bursztein, D. Boneh, and C. Jackson. Busting frame busting: a study of clickjacking vulnerabilities at popular sites. In *in IEEE Oakland Web 2.0 Security and Privacy (W2SP 2010)*, 2010.

[13] D. Sandler. Quick explanation of the 'Don't Click' attack. `http://dsandler.org/outgoing/dontclick_orig.html`, Feb. 2009.

[14] P. Stone. Next Generation Clickjacking - New attacks against framed web pages. `http://www.contextis.co.uk/resources/white-papers/clickjacking/Context-Clickjacking_white_paper.pdf` [09.12.2010], April 2010.

[15] Twitter Blog. Clickjacking Blocked. `http://blog.twitter.com/2009/02/clickjacking-blocked.html` [09.12.2010], Februar 2009.

[16] W. Zeller and E. W. Felten. Cross-Site Request Forgeries: Exploitation and Prevention, Oktober 2008.

doi: 10.2313/NET-2011-05-2_11