A policy based approach for the management of Web browser resources to prevent anonymity attacks in Tor

Guillermo Navarro-Arribas and Joaquin Garcia-Alfaro

Abstract Web browsers are becoming the universal interface to reach applications and services related with these systems. Different browsing contexts may be required in order to reach them, e.g., use of VPN tunnels, corporate proxies, anonymisers, etc. By browsing *context* we mean how the user browsers the Web, including mainly the concrete configuration of its browser. When the context of the browser changes, its security requirements also change. In this work, we present the use of authorisation policies to automatise the process of controlling the resources of a Web browser when its context changes. The objective of our proposal is oriented towards easing the adaptation to the security requirements of the new context and enforce them in the browser without the need for user intervention. We present a concrete application of our work as a *plug-in* for the adaption of security requirements in Mozilla/Firefox browser when a context of anonymous navigation through the Tor network is enabled.

1 Introduction

The Web is increasingly becoming a universal interface for the development of all kinds of applications: from traditional electronic banking and electronic mail, to text processors or even elaborated social networks. As the Web is evolving, the surrounding and supporting technologies are becoming more complex. This is specially relevant in applications that enable the interaction with the Web from the client side: the Web browsers. The current complexity of the Web has a direct im-

G. Navarro-Arribas
IIIA - Artificial Intelligence Research Institute, CSIC - Spanish Council for Scientic Research. Campus UAB s/n, 08193 Bellaterra (Catalonia, Spain) e-mail: guille@iiia.csic.es

J. Garcia-Alfaro
UOC - Universitat Oberta de Catalunya, Rambla Poble Nou 156, 08018 Barcelona (Catalonia, Spain), e-mail: joaquin.garcia-alfaro@acm.org

pact on the security of such applications and more precisely in the treatment of its resources. Attacks against Web browsers can compromise the security and privacy of its users. This can have serious consequences given the pervasive presence of this piece of software in, for instance, important critical systems in industries such as health care, banking, government administration, and so on. Let us mention, for instance, the case of H.D. Moore, the lead developer of the Metasploit Project [12]. One of his projects is based on the exploitation of browser misconfiguration, such as permission of Java and JavaScript code when browsing anonymously through the *The second generation Onion Router* (Tor) network [5], with the objective of catching digital pirates and child pornographers [9]. Even if we agree in the legitimacy of these techniques for the discovery of criminals, these same techniques can lead to violations of fair users. For instance, similar techniques were used by Dan Egerstad in November 2007 [10], for capturing sensible information from legitimate Tor users. As a result of these experiments, several government, embassy, NGO, and other corporate user accounts and passwords were reported and disclosed.

We are currently working on the implementation of a contextual XACML [7] policy manager for Web browsers. The main objective of our work is to be able to automatise the management of resources associated with the browser in a dynamic and flexible way. The use and enforcement of different security contexts will also help in adapting the browsers to the security needs of the working environment of a given user. Such an automatism aims to lead to an error-free process in which non-expert users are protected about security and privacy weaknesses due to browser misconfiguration. We present in this article a concrete application of our proposal to adapt the browser security requirements when an anonymous navigation context is in use. By browsing *context* we mean how the user browsers the Web, including mainly the concrete configuration of its browser. We also describe in this work the current development of our proposal as a *plug-in* for the Mozilla/Firefox family of Web browsers. We consider that our approach must be seen as a design recommendation for future applications dealing with the Web paradigm.

2 Overview of the proposal and plan of the paper

The article is organised as follows. In this Sec. 2 we introduce the XACML language, and the development of our proposal as a *plug-in* for Mozilla/Firefox browsers. In Sec. 3 we show a concrete application of our proposal to adapt the security requirements of Mozilla/Firefox to anonymous Web browsing through the Tor project infrastructure. We conclude the article in Sec. 4.

2.1 XACML

XACML (*eXtensible Access Control Markup Language*) is an XML based standard language [7], which provides the ability to specify both the access control policy and the request/response messages.

In XACML, an access control policy presents an specific format, having as the main element the *rule*. Each rule has an associated *target*, which determines to what (or who) the rule is applied, an *effect*, which is normally *permit* or *deny*, and a condition. If the condition is evaluated in a favourable manner, the result of the evaluation of the rule is the one determined by its effect. One or more rules are associated to a *policy*, which also can specify a target and *obligations*. Such obligations specify actions to be performed by the policy verifier when the policy is applied [16] (normally, these actions will be performed by a Policy Enforcement Point, e.g., a web browser enforcement agent). Finally, one or more policies are included in a *policy set* which can also have an associated target and obligations.

In XACML, the combination of the results of evaluating the rules included in the same policy and the evaluation of the policies included in the same policy set, is given by the combining algorithms. Such algorithms are not only used for the combination of rules and policies, but also for conflict resolution, because they are used when more than one rule or policy is applicable to the same *target*. There is a set of standard algorithms applied both to the combination of rules and the combination of policies. Among them, we remark the following ones:

- *deny-overrides*: an evaluation with *deny* effect takes precedence over the rest.
- *permit-overrides*: an evaluation with *permit* effect takes precedence over the rest.

In our case, in a very summarised way, by using XACML, we can specify the traditional tuple ‘subject-resource-action’ adapted to our concrete problem and context. That is, specify if a given script (subject) is allowed or not to access and/or modify (action) a given browser resource (object). In Sec. 3.3 we show with more detail how are the policies of our proposal defined.

2.2 The Plug-in for Mozilla/Firefox of our proposal

The specific implementation of our authorisation proposal, from now on XAPO (*XAcml Policy Officer*), is based on the Mozilla development framework for the implementation of browser extensions (plug-ins) in the Mozilla/Firefox Web browser. The development of XAPO is mainly based on Java, JavaScript, and XUL (*XML User Interface Language*) [6]. The plug-in is executed in the browser through the *chrome* interface used by the Mozilla applications [11]. From this interface, XAPO, as any other code executed in *chrome* mode, can perform the actions required by our proposal such as access to configuration options, storage and reading preferences, or activate and deactivate browser components (i.e Java, JavaScript, or Shockwave/Flash, etc.). This is done through the XPCOM interface of the Mozilla/Firefox browser. This option is only available in version 3 of the browser. For the implementation of the XACML components we have used *SunXACML* [18], an open source implementation of the XACML standard in Java. Such implementation, is executed inside XAPO by making use of the *LiveConnect* interface provided by Mozilla. The installation of all the set of components of XAPO is done with a single *xpi* package. The current version of XAPO is available under demand. In the following section we present the use of XAPO to adapt the security requirements of the Mozilla/Firefox browser when an anonymous browsing context is activated.

3 Preventing attacks on a context of anonymous browsing

We present in this section a specific application of our proposal. It allows us to adapt the security requirements of a browser when a context of anonymity is enabled on it. Our example scenario is based on the anonymous infrastructure of the Tor project. We introduce in the following subsection some characteristics of Tor, as well as the specific attack which is going to be addressed by our proposal.

3.1 The anonymity infrastructure of Tor

Several anonymity designs have been proposed in the literature with the objective of hiding senders identities for privacy purposes. From simple proxies to complex systems, anonymity networks can offer either strong anonymity for high latency services (e.g., email and Usenet messages) or weak anonymity for low-latency services (e.g., Web browsing). The most widely-used of the latter solutions is based on anonymous mixes and onion routing [15]. It is distributed as a free software implementation known as *The second generation Onion Router* (*Tor*) [5]. It can be installed as an end-user application on a wide range of operating systems to redirect the traffic of low-latency services with a very acceptable overhead.

The Tor objective is the protection of the anonymity of a sender as well as the contents of its messages. To do so, it transforms cryptographically those messages and mixes them via a circuit of routers. Through this circuit, routers transport the original message in an unpredictable way. The content of each message is moreover re-encrypted within each router with the objective of achieving anonymous communication even if a set of routers are compromised by an attacker. As soon as a router receives a new message, it decrypts its corresponding encryption layer with its private key to obtain the following hop and the encryption key of the following router in the path. This path is initially defined at the beginning of the process. Only the entity that creates the circuit — and which remains at the sender's side during all the process — knows the complete path to deliver a given message. The last router of the path, the *exit* node, decrypts the last layer and delivers an unencrypted version of the message to its target.

The maturity of the project and its low impact to the performance of on-line services make the infrastructure of Tor a promising solution to anonymously browse on Internet. To obtain this low impact over the performance of the services tunnelled by Tor, it relies on a very pragmatic threat model. Such a model assumes that adversaries can compromise some fraction of the onion routers in the network. If so, adversaries can not only observe but also manipulate some fraction of the network traffic of Tor. A first implication of this assumption is that the exit node has a complete view of the sender's messages. Therefore, without other countermeasures, it could perform a *Man-in-the-Middle* attack to forge answers. As a result, a malicious onion router acting as the exit router could try to redirect the client to malicious services or to perform denial of service. A second implication of the threat model of Tor is the possibility of suffering traffic analysis attacks with the objective of tracing back the sender's origin or to degrade Tor's anonymity. Several traffic analysis attacks against Tor have been reported in the literature, such as [2, 13, 19].

A third problem raises when the configuration of a browser is not handled properly. Beyond the proper installation and configuration of the software downloaded from the Tor project, some aspects of the browser must be adapted. Anonymous browsing with Tor requires not only different habits, but also reconfiguration of some resources. It is necessary to disable, for example, the execution of JavaScript and Java code, as well as plug-ins like Flash, ActiveX, etc. The use of cookies associated with previous visited sites, on the other hand, must also be taken into account. It might be relatively simple for an attacker to manipulate these components in order to obtain the identity or location of the user (e.g., by obtaining a public IP address associated with the user). We show in the following subsection a practical example that shows how to obtain the IP address of a browser configured to browse through the Tor network. The attack exploits a misconfigured browser that allows the execution of Java code.

3.2 Bypassing Tor via attacks targeting Web browsers

In order to browse through the network of Tor, users should first configure their browsers to redirect its requests and responses via an HTTP proxy, such as Privoxy [14]. In fact, not only HTTP traffic must be redirected by the proxy. Any other traffic, such as DNS requests and responses, must be redirected. Privoxy and Tor allow these later redirections through the use of the SOCKS protocol [8]. There are many other resources on the browser that could leak information if they are not redirected by the proxy. The large amount of options on current browsers leads to an error prone process. The activation and execution of code by plug-ins, such as Flash, Java, ActiveX, etc., increases the dynamism of Web services, but also increases the number of potential targets to exploit. If these resources are not properly managed, an attacker can get control of them and violate user's anonymity via covered channels.

In [1], Abbott et al. describe the use of this kind of attacks, executed within Web browsers, in order to bypass the anonymity of Tor. Forcing the user to visit a specific

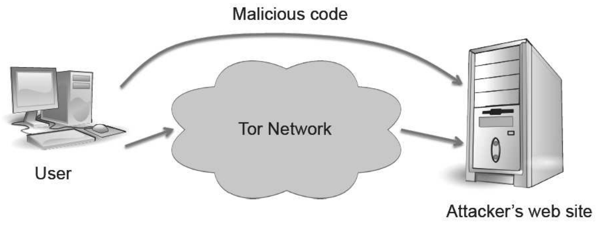


Fig. 1 Example of a Web attack to bypass the anonymity of Tor.

Web site, e.g., using social engineering, phishing, or *Man-in-the-Middle* attacks, a malicious code embedded within the pages of such service opens a secret channel between the user and the attacker's Web domain. Later, performing an analysis of the traffic exchanged with each victim, the attacker collects and stores data related with the resources of each browser (e.g., IP addresses, operating system, browser characteristics, etc.). It is important to note that the collection of this information is not indeed an attack against Tor's infrastructure (cf. Fig. 1). The attack relies on the exploitation of tools and browser runtime components. More specifically, the attack is exploiting browser misconfiguration to bypass its proxy settings.

In [4], Christensen et al. extend this previous attack in order to compromise the identity of Tor users without the necessity of controlling end services (i.e., the visited Web service). The attacker only needs to control exit nodes of Tor. From these nodes, and modifying HTTP traffic, the attacker can successfully execute a *Man-in-the-Middle* attack to reveal user and hidden service identities. The modification of HTTP traffic aims at marking the traffic. For example, the use of HTML elements of type *iframe*, can allow the attacker to include unique references leading to malicious Web sites, as well as to associate a specific *cookie* to collect user data. This reference can force the browser to download malicious code, such as Java or Flash code. If the plug-in that is required by such code is enabled, the code can manage to steal user information and direct the output towards the attacker. Similarly to the attack shown in Fig. 1, the attacker can post-process the information in order to perform an analysis of traffic trying to reveal the identity and activities of the set of victim users. Abbot et al. show in [1] how this and other similar attacks can be extended in order to increase the chance of discovery of Tor users and hidden services.

3.3 Using XAPO and XACML policies to prevent the attack

To prevent attacks against the anonymity provided by Tor as the one described in Sec. 3.2, we use a concrete type of policy, which allows not only to prevent such attacks but also to introduce enough flexibility and fine-grained specification to be adapted to several contexts and degrees of anonymity.

The XACML policy used is divided in two specific policies. On one hand there is a general policy, which explicitly determines the browser resources that have to be protected: Java, JavaScript, . . . and on the other hand there is a *whitelist*-like policy that provides a fine-grained control of the trusted domains for which the activation and/or access to concrete resources is allowed.

The first policy is the *generic-tor-policy*. It is composed of a *policy* element containing a rule for each browser resource to be protected. The effect of such rules is always *deny*, indicating that such resource cannot be accessed when the policy is enforced (c.f. Fig. 2).

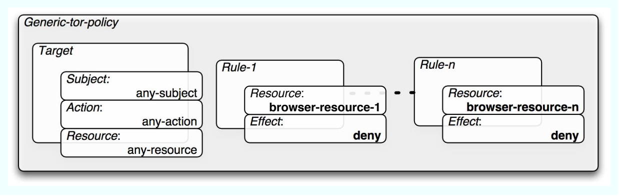


Fig. 2 Generic-tor-policy.

The main purpose of the *generic-tor-policy* is to globally avoid problems such as the one described in Sec. 3.2. To that end, the access to all sensitive resources is explicitly denied when Tor is in use. Some important resources that need to be protected are [1]:

- Browser plug-ins such as: Java, Flash, ActiveX, RealPlayer, Quicktime, Adobe PDF, ... One can specify in the policy plug-ins one by one or use the special resource *all-plugins*. With this last reference, XAPO looks all the plug-ins currently installed in the browsers and turns them off.
- Cookies: it is important to protect the access to cookies, which could have been created previously to the activation of the Tor navigation.

As it can be appreciated, this policy is very restrictive and can limit the functionality of the applications accessed by the user. In order to improve the user experience, we consider it important to provide a *whitelist*-like policy to allow the definition of trusted domains, which are allowed to access some browser resources. This avoids the common scenario where a user is using two different browsers, one with Tor activated and with a minimal functionality and another one without Tor and with a compete or extended functionality. That is, the user can determine some trusted applications and allow them to access given resources without giving up the anonymity measures provided by Tor and XAPO in the other domains.

The *tor-whitelist-policy* defines the domains which are allowed to access concrete browser resources. For each trusted domain, there is an specific policy, which has rules to describe which actions are allowed over which resources. The effect of

[1] The policy may include other needed resources a part from plug-ins and cookies.

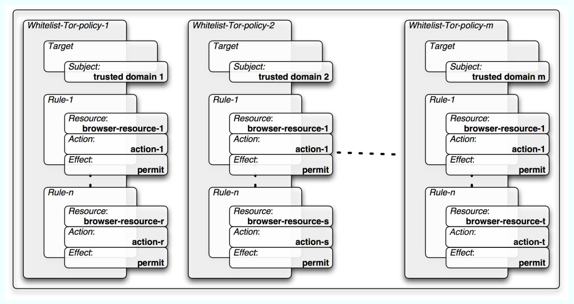


Fig. 3 Tor whitelist policy.

these rules is *permit* and it will have preference over the evaluation of the *generic-tor-policy* (c.f. Fig. 3). Through XAPO, the user can choose the trusted domains and enable all the desired browser options and resources for them. This changes are stored in the corresponding whitelist policy and will take effect for the successive executions of the browser.

Both the generic policy and the whitelist policy are combined in an XACML policy set by the *permit-overrides* policy combining algorithm (c.f. Sec. 2.1). This makes the whitelist policy to take precedence over the generic one. Or in other words, the whitelist policy expresses *exceptions* of the generic policy.

3.4 Example of XAPO policies for Tor

In this section we will show a simple example of XAPO policies for Tor in order to ensure an additional enforcement level and to prevent attacks against the anonymity of users even when they are browsing with Tor enabled. The description of the policies has been simplified to improve its legibility. At the same time, the example is quite simple, but it shows clearly and concisely, the way these policies work.

In the following listing (Listing 1) we show an example of a *generic-tor-policy*. The policy includes three rules: *java-plugin*, *javascript-plugin*, *cookies*. The first one makes XAPO to disable the Java plug-in, the second ones disables the JavaScript interpreter, and the third one prevents the reading of cookies for all domains.

```
<Policy PolicyId=”tor−generic:default−tor−firefox”
        RuleCombiningAlgId=”deny−overrides”>
  <Target> ... </Target>
  <Rule RuleId=”java−plugin” Effect=”Deny”>
    <Target>
      ...
      <Resources>
        <Resource>
          <ResourceMatch
              MatchId=”function:anyURI−equal”>
            <AttributeValue DataType=”XMLSchema \
              #anyURI”>urn:browser:plugin:java
            </AttributeValue>
            <ResourceAttributeDesignator
                DataType=”XMLSchema#anyURI”
                AttributeId =”resource:resource−id”/>
          </ResourceMatch>
        </Resource>
      </Resources>
    </Target>
  </Rule>
  <Rule RuleId=”javascrip−plugin” Effect=”Deny”>
    <Target>
      ...
      <Resources>
        <Resource>
          <ResourceMatch MatchId=”function:\
                                   anyURI−equal”>
            <AttributeValue DataType=”XMLSchema\
                                       #anyURI”>
              urn:browser:plugin:javascript
            </AttributeValue>
            <ResourceAttributeDesignator
                DataType=”XMLSchema#anyURI”
                AttributeId =”resource:resource−id”/>
          </ResourceMatch>
        </Resource>
      </Resources>
    </Target>
  </Rule>
  <Rule RuleId=”cookies” Effect=”Deny”>
    <Target>
      <Subjects><AnySubject/></Subjects>
      <Actions>
        <Attribute  AttributeId =”action:action −id”
                    DataType=”XMLSchema#string”>
          <AttributeValue>read</AttributeValue>
        </Attribute>
      </Actions>
      <Resources>
        <Resource>
          <ResourceMatch
              MatchId=”function:anyURI−equal”>
            <AttributeValue
                DataType=”XMLSchema#anyURI”>
              urn:browser:document.cookie
            </AttributeValue>
            <ResourceAttributeDesignator
                DataType=”XMLSchema#anyURI”
                AttributeId =”resource:resource−id”/>
          </ResourceMatch>
        </Resource>
      </Resources>
    </Target>
  </Rule>
</Policy>
```

Listing 1 Generic policy for Tor.

Following the example, the user may want to activate the JavaScript interpreter and the Java plug-in but just for a concrete trusted email Web application (`mail.trusted.domain.org`), which is accessed through HTTPS. Instead of having to change or disable the Tor generic policy or initiate a new session without Tor, the user can include a whitelist policy to make XAPO allow the execution of JavaScript code from the trusted domain. The following listing (Listing 2) shows a policy to apply the corresponding domain with two rules, one to activate the Java plug-in and another for JavaScript.

```
<Policy PolicyId=”tor− whitelist:mail ”
        RuleCombiningAlgId=”permit−overrides”>
  <Target>
    <Subjects>
      <Attribute  AttributeId =”subject:subject −id”
                  DataType=”XMLSchema#anyURI”>
        <AttributeValue>
          https: // mail.trusted .domain.org
        </AttributeValue>
      </Attribute>
    </Subjects>
    ...
  </Target>
  <Rule RuleId=”java−rule” Effect=”Permit”>
    <Target>
      ...
      <Resources>
        <Resource>
          <ResourceMatch
              MatchId=”function:anyURI−equal”>
            <AttributeValue DataType=”XMLSchema\
              #anyURI”>urn:browser:plugin:java
            </AttributeValue>
            <ResourceAttributeDesignator
                DataType=”XMLSchema#anyURI”
                AttributeId =”resource:resource−id”/>
          </ResourceMatch>
        </Resource>
      </Resources>
    </Target>
  </Rule>
  <Rule RuleId=”javascrip−rule” Effect=”Permit”>
    <Target>
      ...
      <Resources>
        <Resource>
          <ResourceMatch
              MatchId=”function:anyURI−equal”>
            <AttributeValue DataType=”XMLSchema\
            #anyURI”>urn:browser:plugin:javascript
            </AttributeValue>
            <ResourceAttributeDesignator
                DataType=”XMLSchema#anyURI”
                AttributeId =”resource:resource−id”/>
          </ResourceMatch>
        </Resource>
      </Resources>
    </Target>
  </Rule>
</Policy>
```

Listing 2 Tor-whitelist policy for the domain trusted.domain.org.

Finally, in the following listing we show another whitelist policy (Listing 3), which tells XAPO to enable JavaScript only for the domain `trusted-bank.org`.

```
<Policy PolicyId=”tor−whitelist:bank”
        RuleCombiningAlgId=”permit−overrides”>
  <Target>
    <Subjects>
      <Attribute  AttributeId =”subject:subject−id”
                  DataType=”XMLSchema#anyURI”>
        <AttributeValue>trusted−bank.org</AttributeValue>
      </Attribute>
    </Subjects>
    ...
  </Target>
  <Rule RuleId=”javascrip−rule” Effect=”Permit”>
    <Target>
      <Subjects><AnySubject/></Subjects>
      <Actions><AnyAction/></Actions>
      <Resources>
        <Resource>
          <ResourceMatch
            MatchId=”function:anyURI−equal”>
            <AttributeValue
              DataType=”XMLSchema#anyURI”>
              browser:plugin:javascript
            </AttributeValue>
            <ResourceAttributeDesignator
                DataType=”XMLSchema#anyURI”
                AttributeId =”resource:resource−id”/>
          </ResourceMatch>
        </Resource>
      </Resources>
    </Target>
  </Rule>
</Policy>
```

Listing 3 Tor-whitelist policy for the domain trusted-bank.org.

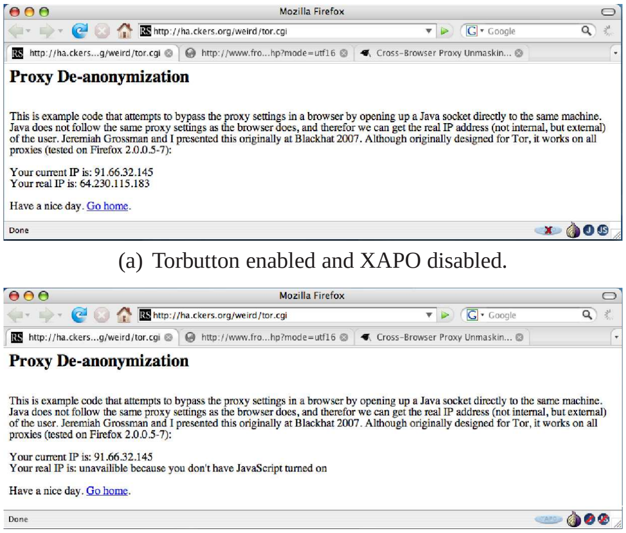
As it can be seen, the policy allows the activation of the JavaScript interpreter for the concrete domain through the corresponding rule.

The three policies we have seen in the example, are combined in a policy set. A simplified example of such policy set can be seen in the following listing (Listing 4).

```
<PolicySet PolicySetId=”xapo:tor−policyset”
           PolicyCombiningAlgId=”permit−overrides”>
  <Target />
  <PolicyIdReference>
    tor−generic:default−tor−firefox
  </PolicyIdReference>
  <PolicyIdReference>
    tor−whitelist:mail
  </PolicyIdReference>
  <PolicyIdReference>
    tor−whitelist:bank
  </PolicyIdReference>
</PolicySet>
```

Listing 4 Policy set example for Tor.

To conclude this section, we show with the practical example presented in Figs. 4(a) and 4(b), the way the activation of XAPO prevents the attacks seen in Sec. 3.2. The browser used is the Mozilla/Firefox 3 Beta 1, configured with XAPO and the Torbutton extension [17] (used for the automatic configuration of Privoxy in the browsing preferences of Mozilla/Firefox). As it has already been discussed, the elevated number of configuration options present in a Mozilla/Firefox browser make it possible, without the proper measures, to third parties to violate the anonymous channels provided by Tor and retrieve without problems the identity of the browser. The attack shown exploits the use of a Java code executed from JavaScript in order to open a socket through *LiveConnect*. This code makes an HTTP request to the server hosting the web page (`http://ha.ckers.org/weird/tor.cgi`). Given that the request does not go through the nodes of the Tor network, after a simple analysis of the received request, and automatically, the attacker of the visited web site gets to know and shows in the screen information associated to the user, such as the IP address. Fig. 4(b) shows how the activation of XAPO and thus the protections of resources associated with the XAPO policies, prevents the creation of the channel between the attacker and the victim browser.



Mozilla Firefox
http://ha.ckers.org/weird/tor.cgi
Google
http://ha.ckers...g/weird/tor.cgi
http://www.fro...hp?mode=utf16
Cross-Browser Proxy Unmaskin...
Done

Proxy De-anonymization

This is example code that attempts to bypass the proxy settings in a browser by opening up a Java socket directly to the same machine. Java does not follow the same proxy settings as the browser does, and therefor we can get the real IP address (not internal, but external) of the user. Jeremiah Grossman and I presented this originally at Blackhat 2007. Although originally designed for Tor, it works on all proxies (tested on Firefox 2.0.0.5-7):

Your current IP is: 91.66.32.145
Your real IP is: 64.230.115.183

Have a nice day. Go home.

(a) Torbutton enabled and XAPO disabled.

Mozilla Firefox
http://ha.ckers.org/weird/tor.cgi
Google
http://ha.ckers...g/weird/tor.cgi
http://www.fro...hp?mode=utf16
Cross-Browser Proxy Unmaskin...
Done
XAPO

Proxy De-anonymization

This is example code that attempts to bypass the proxy settings in a browser by opening up a Java socket directly to the same machine. Java does not follow the same proxy settings as the browser does, and therefor we can get the real IP address (not internal, but external) of the user. Jeremiah Grossman and I presented this originally at Blackhat 2007. Although originally designed for Tor, it works on all proxies (tested on Firefox 2.0.0.5-7):

Your current IP is: 91.66.32.145
Your real IP is: unavailible because you don't have JavaScript turned on

Have a nice day. Go home.

(b) Torbutton and XAPO enabled.

Fig. 4 (a): Example of an attack to bypass the configuration of Privoxy in a Mozilla/Firefox browser with the Torbutton extension enabled and the XAPO extension disabled. The attack opens an addition channel between the execution environment of the browser and the attacker and, through this channel, it extracts the information associated with the browser; (b): Prevention of the attack by enabling the XAPO extension.

4 Conclusions

In this article we have presented a proposal to apply security policies in a Web browser. More precisely, we have presented a Mozilla/Firefox extension, which allows the use of policies expressed in the XACML standard language to protect the resources of the browser. Furthermore, we have shown how this extension, named XAPO, can be used to enhance the anonymity of the users as a complement to the network infrastructure of the Tor project.

Tor suffers some security problems, since there are already known attacks, which can violate the anonymity of its users. By using a malicious code based on, for example Java or Flash animations, an attacker can set up a direct connection with Web servers under its control and the browser, jeopardising the anonymity of the user. Although the attack is actually exploiting the tools and the external environment of the Tor network, there are attacks in the literature (see, for instance, [1]) showing how to extend this and other similar attacks with the aim of augmenting the probabilities of an attacker to violate the anonymity of the users and services hided behind the network of the Tor project.

Our proposal allows to prevent such attacks by defining an enhanced security policy oriented to Tor, which guarantees to the user a better protection of his identity and sensitive information. To that end, the policy allows the definition of the browser resources that have to be protected as an additional measure to the protection already provided by Tor. Such policy is flexible enough to be adapted to all the browsing habits of the user. For example, while it provides a complete protection, it also allows the definition of a *whitelist* of trusted domains, which are allowed to use some given resources, improving the browsing experience of the user

Currently, the prototype of the proposal is being developed as an extension of the Mozilla/Firefox browser, but we are working on the development of equivalent extension for other browser such as Safari, or Internet Explorer. The use of an standard language such as XACML, allows to easily reuse and interchange the policies between different browsers.

Acknowledgements Partial support by the Spanish MEC (projects eAEGIS TSI2007-65406-C03-02, and ARES - CONSOLIDER INGENIO 2010 CSD2007-00004) is acknowledged.

References

1. Abbott, T., Lai, K., Lieberman, M., and Price, E. Browser-Based Attacks on Tor. *7th Workshop on Privacy Enhancing Technologies (PET 2007)*, LNCS 4776(1):184–199, 2007.
2. Bauer, K., McCoy, D., Grunwald, D., Kohno, T., and Sicker, D. Low-resource routing attacks against Tor. *ACM workshop on Privacy in electronic society*, pp. 11–20, 2007.
3. Chaum, D. Untraceable electronic mail, return addresses, and digital pseudonyms. *In: Communications of the ACM*, 24(2):84–88, 1981.
4. Christensen, A. et al. Practical Onion Hacking.. *FortConsult*, October 2006.
5. Dingledine, R., Mathewson, N., and Syverson, P. F. Tor: The second-generation Onion Router. *In: 13th conference on USENIX Security Symposium*, 2004.
6. Ginda, R. Writing a Mozilla Application with XUL and Javascript. *O'Reilly*, USA, 2000.
7. Godik, S., et al. eXtensible Access Control Markup Language (XACML) Version 2. Standard, OASIS. February 2005.
8. Leech, M., et al. SOCKS Protocol Version 5. *RFC1928*, March 1996.
9. Lemos, R. Tor hack proposed to catch criminals. SecurityFocus, March 2007. [Online]. Available: http://www.securityfocus.com/news/11447
10. Lemos, R. Embassy leaks highlight pitfalls of Tor. SecurityFocus, September 2007. [Online]. Available: http://www.securityfocus.com/news/11486
11. Mcfarlane, N. *Rapid Application Development with Mozilla*. Prentice Hall PTR., 2004.
12. Moore, H. D. et al. The Metasploit Project. [Online]. Available: http://www.metasploit.com/
13. Murdoch, S. J. and Danezis, G. Low-cost traffic analysis of Tor. *IEEE Symposium on Security and Privacy)*, pp. 183–195, 2005.
14. Privoxy - Home Page [Online]. Available: http://www.privoxy.org/
15. Reed, M. G., Syverson, P. F., and Goldschlag, D. M. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16(4):482–494, 1998.
16. Sloman, M. Policy Driven Management for Distributed Systems *Journal of Network and Systems Management*, vol. 2, part 4. Plenum Press. 1994.
17. Perry M. and Squires S. Torbutton. [Online]. Available: https://www.torproject.org/torbutton/
18. Sun Microsystems SunXACML. [Online]. Available: http://sunxacml.sourceforge.net
19. Wright, M. K., Adler, M., Levine, B. N., Shields, C. Passive-Logging Attacks Against Anonymous Communications Systems. *ACM Transactions on Information and System Security (TISSEC)*, Vol. 11, No. 2, Article 7, 1–33, Pub. date: May 2008.
20. Yavatkar, R., Pendarakis, D. and Guerin, R. A Framework for Policy-based Admission Control RFC 2753. The Internet Society. January, 2000.