Security Threat Analysis for Hughes' Prefetching Proxy

Brian Jones
Stanford University
March 14, 2011

Abstract

This project, proposed by HughesNet, examines security vulnerabilities associated with using a prefetching proxy. Hughes is interested in using a prefetching proxy to reduce noticeable latency on the user's end while connected to the internet via satellite. This paper looks at three possible threats to this system, security properties associated with these threats, and possible solutions.

Motivation

In satellite communications, there is no avoiding the major issue of high latency. Given that the speed of light isn't changing anytime in the foreseeable future, satellite internet providers must look to different mechanisms for reducing latency. HughesNet's approach to reduce the noticeable latency on the user's end is to implement a prefetching proxy which prefetches embedded content in user requested webpages. The theory behind this approach is the user will get quicker feedback from a website, even though the user may still have to wait for the data or media they were asking for.

System Model

As shown in figure 1, the prefetching proxy, located on the Hughes/Internet side of the satellite connection, passes back the plain HTML immediately to the user after receiving a response from a website. Instead of the user continuously requesting the extra embedded objects in the website, the prefetching proxy makes the requests and forwards the objects back to the user once it receives them.

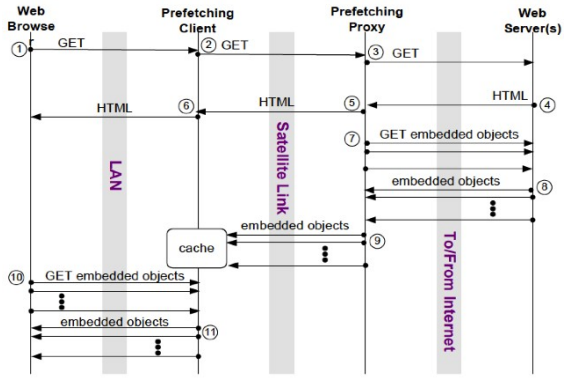

Figure 1: Prefetching Proxy System Model

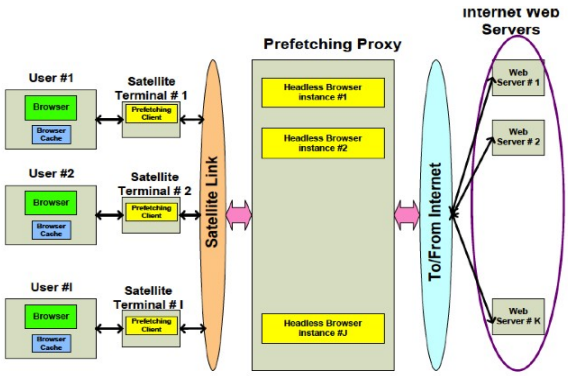

Figure 2: Prefetching Proxy System Model

[1]

The structure of the prefetching proxy requires some discussion. The prefetching proxy consists of a set of "headless browsers." Each instance of a user's browser is assigned a headless browser in the prefetching proxy. These act on the users's browser's behalf in communication with user requested websites. The creation and deletion of these headless browsers is linked directly with the browsers loaded and unloaded on the user's computer.

The original model from Hughes kept the

1 Pictures courtesy of HughesNet

headless browsers strictly disparate entities. While this constraint kept the model highly secure, it also limited performance. With the structure of the prefetching proxy proposed, performance could be drastically improved by allowing headless browsers to share data requested by different users. Much more care has to be taken in implementing this type of system, but the possible improved performance warrants the efforts. For this reason, I have used a shared cache in the prefetching proxy in my model and I analyze some of the security vulnerabilities related to using a shared cache.

Threat Model

In this paper I examine three threat models. Two of these are related to the use of a shared cache and the third is related to an SSL_Strip attack, independent of the shared cache.

The first threat I look at is an adversary having access to the prefetching proxy and the shared cache with normal user permissions. Such an adversary can eavesdrop on traffic moving through the prefetching proxy by requesting only data stored in the shared cache. This type of threat threatens a user's confidentiality by possibly sharing sensitive user information with an adversary.

The second threat I look at is an attacker's website being visited by a user and the website planting malicious code onto the prefetching proxy. As a result, the malicious code is stored in the shared cache and can then be planted on all the other users communicating with the prefetching proxy. This type of threat threatens the integrity of the system due to the possible vastness of the infection. With such a potentially detrimental infection, the attacker could disrupt the normal operation of the system.

The third threat I look at is an HTTPS SSL_Strip attack. This type of attack could be quite devastating for the user's privacy, even without the exploitation of the shared cache. With a capable adversary, a user's secure HTTPS session could be hijacked, giving the user's credentials for the HTTPS session to the attacker. Despite this crippling attack, there could be a defense against it by using the prefetching proxy to help monitor messages passing through it.

Security Properties

Here I outline the security properties related to the threat models described above.

Confidentiality
We want to prevent an adversary from monitoring traffic through the prefetching proxy. In effect, nonsensitive data should be shared between the headless browsers, but no browser should know the intended user of the data.

Integrity
Malware downloaded to the prefetching proxy should be prevented from being shared and should not reach the end users.

Privacy
HTTPS transfers should be trusted to pass through the prefetching proxy securely.

Modeling in Alloy

My tool/language of choice to model and verify these security properties was Alloy. I chose Alloy since it lends itself to describe such abstract systems. Provided that I'm modeling a general system and not a specific protocol, Alloy was naturally a more suitable choice than other description languages such as Murphi.

I successfully modeled the proposed system with shared cache in Alloy. In the process of creating my model I made the following assumptions:

- Users and prefetching clients have a one-to-one mapping.
- One headless browser per user.
- Shared cache holds HTML responses and embedded objects.
- Two different GETs with same content go to the same website.
- Two different GETs with different content go to different websites.

While the model does use some standard messages such as GET requests and HTML responses, it uses these in a very general sense and these are only meant to be used for conceptual purposes. The messages are simply intended to show the flow through the system. The overall system is only meant to aid current protocols; it does not add any new protocols. Constructing my model in such a general manner proved to be less fruitful in terms of discovering vulnerabilities through the use of Alloy. As a result, I analyzed all of my proposed vulnerabilities by inspection. Given more time, I would choose to focus on a specific protocol to analyze in this system.

Threat Analysis

Through my use of Alloy, I was able to construct my first two threat models; both of which were confirmed. This, and the modeling of the underlying system was the extent of my use of Alloy. I continue with my analysis through inspection.

I start my analysis with a picture of an ideal use of the system, as can be seen in figure 3.

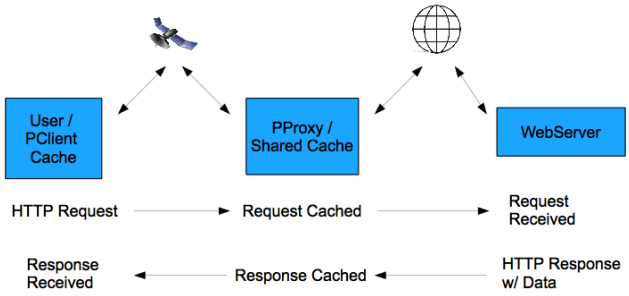

Figure 3: Ideal System

Eavesdropping Attack

An eavesdropping attack, (figure 4), is quite possible in this system if the attacker only asks for responses from the shared cache. This can be done by setting the **cache-control** field in the header of a request to **only-if-cached**. As a result, the attacker's requests will not pollute the shared cache with responses and the attacker can monitor the shared cache by brute force.

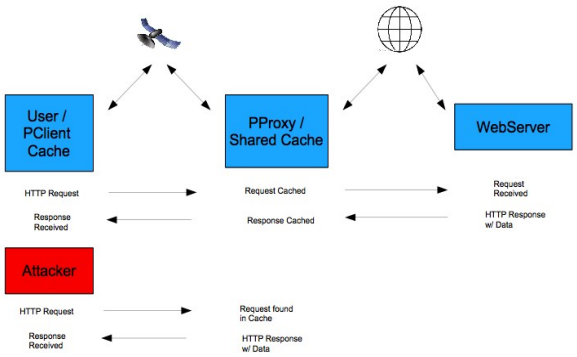

Figure 4: Eavesdropping Attack

Eavesdropping Prevention

A method for preventing this attack consists of modifying select HTTP header fields, (e.g. cache-control), for requests and responses at the prefetching proxy. In effect, the prefetching proxy would treat all HTTP requests the same and all HTTP responses the same. As a result, the attacker would be left unable to determine whether the response came from the shared cache or the website requested. In fact, it is very likely that much of the shared cache would be polluted by the attacker due to its brute force strategy.

Malware Infection

Another attack related to the use of the shared cache is malware infection. It is possible for a malicious website, visited by a user, to plant malware into the shared cache in the prefetching proxy. This can result in collateral damage to other users using the prefetching proxy.

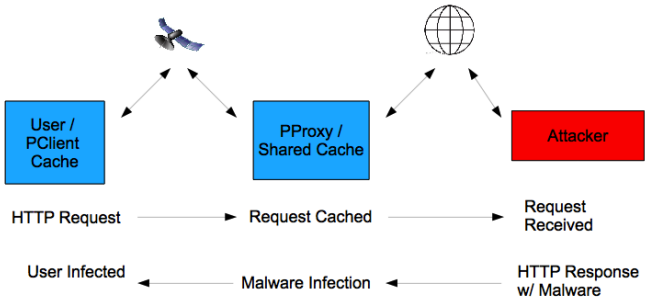

Figure 5: Malware Infection

Malware Prevention

One way to alleviate this problem is to install anti-malware software on the prefetching proxy. This method is not trivial and should be implemented with much care. Another obvious solution is to

not share cache at the prefetching proxy.

HTTPS SSL_Strip Attack

The proposed prefetching proxy system requires SSL bridging. This requires the end user to trust and import the certificate of the SSL bridging server, (the prefetching proxy). This, in itself, raises security concerns for the user since it requires the user to entrust Hughes entirely with all their secure HTTPS connections.

An additional concern is that of an SSL_Strip attack, (figure 7).

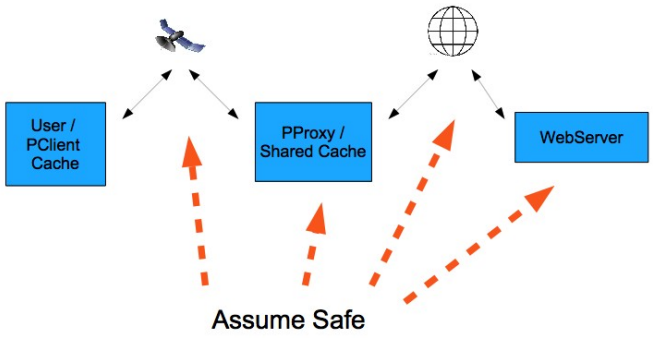

Figure 6: HTTPS SSL_Strip Assumptions

In examining this threat, we are assuming the following entities are safe:

- WebServer – website visited is not malicious
- Internet – attacker on internet cannot determine if traffic is from prefetching proxy
- Prefetching Proxy – no vulnerabilities within the prefetching proxy
- Satellite Connection – encrypted connection is secure

Given these assumptions, I will only look at the connection between the user and the satellite connection.

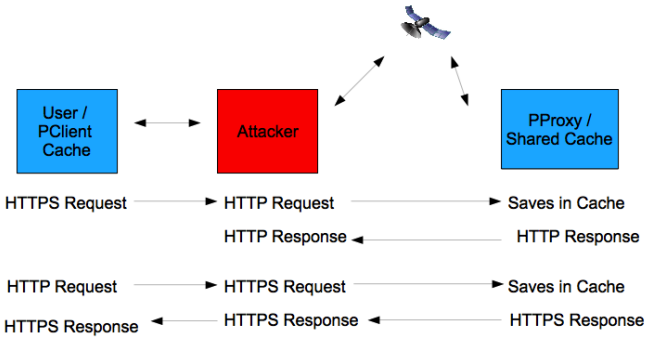

Figure 7: HTTPS SSL_Strip Attack

HTTPS SSL_Strip Prevention

The prefetching proxy provides a convenient intermediate point where we can check messages passing over the system. It is possible for the prefetching proxy to keep track of HTTP messages and HTTPS messages passing through it. If the prefetching proxy sees an HTTP request before an HTTPS request for the same domain name, it can ignore the HTTPS request in suspicion of it being a fraudulent request. In the implementation of this fix, the prefetching proxy would need to keep track of requests per user. This is needed to reduce the number of incorrect suspicions. For example, we don't want to be suspicious of an HTTPS request when the prior HTTP request was from a different user.

Conclusion

Through my analysis, I learned that it is very difficult to analyze a general system without a particular protocol in mind. Provided more time I would further the study one of two ways. Either I would iterate over many different protocols, analyzing each one individually, or I would make my Alloy model much more flexible in terms of messages passing from entity to entity. One way to do the latter could be to assign two message endpoints to each message rather than lay out every intermediate node touched by the message.

Each threat analyzed relates to a different class of security property. As seen in the security properties section, the eavesdropping attack threatens confidentiality, the malware attack threatens integrity, and the HTTPS SSL_Strip attack threatens privacy.

Related Articles

Smart CDNs
http://w2spconf.com/2010/papers/p13.pdf

DNS Prefetching
http://blog.chromium.org/2008/09/dns-prefetching-or-pre-resolving.html

SSL_Strip
http://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf

Prefetching Proxy Implementation
http://www.isoc.org/inet97/proceedings/A1/A1_3.HTM