

[REVS] Hold Your Sessions: An Attack on Java Session-id Generation

Source: <http://www.derkeiler.com/Mailing-Lists/Securiteam/2005-02/0047.html>

From: SecuriTeam (support_at_securiteam.com)

Date: 02/13/05

To: list@securiteam.com

Date: 13 Feb 2005 18:57:53 +0200

The following security advisory is sent to the securiteam mailing list, and can be found at the SecuriTeam web site: <http://www.securiteam.com>

-- promotion

The SecuriTeam alerts list – Free, Accurate, Independent.

Get your security news from a reliable source.

<http://www.securiteam.com/maillinglist.html>

Hold Your Sessions: An Attack on Java Session-id Generation

SUMMARY

HTTP session-id s take an important role in almost any web site today. This paper presents a cryptanalysis of Java Servlet 128-bit session-id s and an efficient practical prediction algorithm. Using this attack an adversary may impersonate a legitimate client. Through the analysis we also present a novel, general space-time tradeoff for secure pseudo random number generator attacks.

DETAILS

Introduction:

At the root of many security protocols, one finds a secret seed which is supposedly generated at random. Unfortunately, truly random bits are hard to come by, and as a consequence, often security hinges on shaky, low entropy sources. In this paper, we reveal such a weakness in an important e-commerce building block, the Java Servlets engine.

Servlets generate a session-id token which consists of 128 hashed bits and must be unpredictable. Nevertheless, this paper demonstrates that this is not the case, and in fact it is feasible to hijack client sessions, using

a few legitimately obtained session-ids and moderate computing resources.

Beyond the practical implication to the thousands [16] of servers using Servlets, this paper has an important role in describing an attack on a pseudorandom-number-generator (PRNG) based security algorithm and in demonstrating a nontrivial reverse engineering procedure. Both can be used beyond the Servlets attack described henceforth.

Web server communication with clients (browsers) often requires state. This enables a server to remember the client's already visited pages, language preferences, shopping basket and any other session or multi-session parameters. As HTTP [9] is stateless, these sites need a way to maintain state over a stateless protocol. Section 2 describes various alternatives for implementing state over HTTP. However, the common ground of all these schemes is a token traversing between the server and the client, the session-id.

The session-id is supported by all server-side frameworks, be it ASP, ASP.net, PHP, Delphi, Java or old CGI programming. Session-ids are essentially a random value, whose security hinges solely on the difficulty of predicting valid session id's. HTTP session hijacking is the act where an adversary is able to conduct a session with the web server and pretend to be the session originator. In most cases, the session-ids are the only means of recognizing a subscribing client returning to a site. Therefore, guessing the unique session-id of a client suffices to act on its behalf.

Driven by this single point of security, we set out to investigate the security of session-id deployments, and as our first target, we have analyzed the generation of session-ids by Apache Tomcat. Apache [2] is an open-source software projects community. The Apache web server is the foundation's main project. According to Netcraft [16] web study of more than 48,000,000 web servers, the Apache web server is used by more than 67% of the servers and hence the most popular web server for almost a decade. At the time of this writing (April 2004), sites such as www.nationalcar.com, www.reuters.com, www.kodak.com and ieeexplore.ieee.org are using Java Servlets on their production web sites.

In many of these sites, the procedure for an actual credit-card purchase requires a secure TLS [8] sessions, separated from the "browsing and selection" session. However, this is not always the case. For example, Amazon's patented [10] one-click checkout option permits subscribing customers to perform a purchase transaction within their normal browsing session. In this case, the server uses a client's credit-card details already stored at the server, and debits it based solely on their session-id identification.

In either of these scenarios, an attacker that can guess a valid client id can easily hijack the client's session. At the very least, it can obtain client profile data such as personal preferences. In the case of a subscriber to a sensitive service such as Amazon's "one-click", it can order merchandise on behalf of a hijacked client. Briefly, our study of

Securiteam: [REVS] Hold Your Sessions: An Attack on Java Session-id Generation

the generation of Java Servlets session-ids reveals the following procedure. A session-id is obtained by taking an MD5 hash over 128-bits generated using one of Java's pseudo-random number generators (PRNG).

Therefore, two attacks can be ruled out right away. First, a brute force search of valid session-ids on a space of 2^{128} is clearly infeasible. Second, various attacks on PRNGs, e.g., Boyar's [6] attack on linear congruential generators, fail because PRNG values are hidden from an observer by the MD5 hashing. Nevertheless, we are able to mount two concrete attacks. We first show a general space-time attack on any PRNG whose internal state is reasonably small, e.g., 264-280. Our attack is resilient to any further transformation of the PRNG values, such as the above MD5 hashing. Using this attack, we are able to guess session-ids of those Servlets that use the `java.util.Random` package, whose internal PRNG state is 64-bits. Beyond that, our generic PRNG attack is the first to use space-time tradeoffs, and may be of independent interest. Our second attack is on the seed-generation algorithm of Java Servlets. Using intricate reverse engineering, we show a feasible bound for the seed's entropy. Consequently, we are able to guess valid session-ids even when Servlets are using the `java.security.SecureRandom` secure PRNG (whose internal state is 160 bits).

The paper is organized as follows. In Section 2 we describe the HTTP state mechanisms. In Section 3 we describe and analyze the Tomcat session-id generation algorithm. Java `hashCode()` study is presented in Section 4. In section 5 we present our attacks on the session-id. We conclude in Section 6.

ADDITIONAL INFORMATION

The information has been provided by <<mailto:zvikag@cs.huji.ac.il>>
Zvi Gutterman.

The original article can be found at:

<<http://www.cs.huji.ac.il/~zvikag/ZviGuttermanHomePage.files/GuttermanMalkhi2005.pdf>> Hold Your Sessions: an Attack on Java Servlet Session-id Generation

=====

This bulletin is sent to members of the SecuriTeam mailing list.

To unsubscribe from the list, send mail with an empty subject line and body to:
list-unsubscribe@securiteam.com

In order to subscribe to the mailing list, simply forward this email to: list-subscribe@securiteam.com

=====
=====

DISCLAIMER:

The information in this bulletin is provided "AS IS" without warranty of any kind.

In no event shall we be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages.