5. Additional attacks

So far the discussion was around the ability to recover the Math.random() seed and/or the CRT rand/srand seed. This represents information leakage (when the seed value is e.g. the browser startup time or the first time the random function was invoked) in itself, as well as a means to tell browser instances apart, thereby effectively enabling tracking users. The PRNG mileage can be used to further distinguish among browsers which have the same seed value.

However, there are several other interesting attacks that abuse the same vulnerabilities, mostly cross-site ones.

It should be noted that while there is no explicit requirement for the Math.random() PRNG (or for the boundary string value) to be cryptographically strong, there is still a legitimate expectation that those mechanisms would not leak information (or be influenced) across domains. So while it is probably a bad idea for an application to rely on strong randomness of Math.random(), it is nevertheless a valid assumption (in the author's opinion) that the Math.random() values would not be predictable from **another** domain, nor that they would be in any way influenced by another domain, nor would it be possible to count the number of invocations of Math.random() from another domain.

5.1 Cross-domain application state detection

Suppose a banking site (www.bank.site) has a login page which invokes Math.random() once, and a protected page (i.e. a page that requires the user to be logged-in on order to be accessed) that does not invoke Math.random() at all. In such case, an attacker can dynamically embed (via an IFRAME HTML tag added by Javascript) the protected page in his/her site. The attacker needs to sample Math.random() before and after the IFRAME is embedded. Using the above techniques, the attacker can easily determine whether an additional Math.random() value was consumed. If so, it means that the user was not logged in (because the protected page redirected to the login page, which consumed the additional Math.random value). If not, the user was logged in.

This demonstrates that an attacker can discern among two application states – user logged in, and user not logged in, across domains.

In general, if it is possible to map application states into different amount of Math.random() consumption, then it is possible to discern among those application states across domains.

It should be noted that using Math.random() in web pages is quite popular, e.g. Google Analytics' urchin.js script (http://www.google-analytics.com/urchin.js) invokes Math.random(), and it seems that the standard script for embedding DoubleClick ads uses Math.random() as well.

Another example is (to continue the banking application example above) a post-login page that displays a random security tip for Silver-tier, Gold-tier and Platinum-tier users, additionally a random investment tip for Gold-tier and Platinum tier users, and additionally a random life-style tip for Platinum-tier users only. Suppose the randomization is implemented using Javascript's Math.random(). Then an attacker can dynamically embed this page, and assuming the user is logged in, the attacker can tell the user's tier by the number of Math.random() values consumed (1 – Silver tier, 2 – Gold tier, 3 – Platinum tier).

This works across all browsers studied (as long as an embedded frame is used).

5.2 Cross-domain partial setting of Math.random()

Continuing the previous example, the attacker can engage in a slightly different attack. Let's assume that the attack target is a Gold-tier user, and that the random investment tip is a stock exchange ticker symbol chosen at the client side from a pool of 100 ticker symbols (presumably those researched and recommended by the bank analysts). This would typically be implemented in the following fashion:

```
ticker=ticker_array[Math.floor(Math.random()*100)];
```

Just before the page is rendered and displayed to the user, the attacker can roll forward the Math.random() PRNG, until the next value of Math.floor(Math.random()*100) is any desirable value. The attacker can thus fix the ticker investment tip the user sees to any one of the 100 tickers, per the attacker's whim.

5.3 Cross-domain Math.random() predictability

Another by-product of the research is the ability to predict the next values from Math.random(), and to reconstruct previous values from Math.random(), even across domains. This jeopardizes client-side password generation schemes (google for Javascript password generator, e.g. [28], to get an idea of how widespread this practice is; naturally not all entries indexed by Google are vulnerable, but probably a large part of them is), and similar applications that rely on strong randomness of the Javascript Math.random() facility, or at least on the premise of cross-domain non-leakage.

Math.random() predictability was demonstrated for:

- IE (Windows)
- Firefox (all platforms)
- Safari (Mac OS/X)

Chrome's Math.random() is also predictable, but since it's process scoped, and since Chrome starts a new process in each navigation to a new site, the de-facto scope of Math.random()'s predictability is limited (e.g. two frames on the same page).

It should be noted again that the Javascript standard ([7]) does not require Math.random() to be implemented as a cryptographically strong PRNG. As such, assuming that Math.random() is strong is a web application programming mistake. At the same time, a browser should not leak information about Math.random()'s state, values and seed across domains.

5.4 Cross-site file upload

From the description in section 3, it may appear that forcing the browser to upload arbitrary content as a file to a 3rd party site is impossible. After all, there's no way to instruct the browser which content to upload (save for user assisted attack in which an actual file is chosen by the user).

However, [25] describes a vulnerability is many browsers which is exploited as described in [26] to conduct "cross site file upload" in which a browser bug is used to spoof headers and file content inside a single "part" of the POST request body. There is one tiny shortcoming of the method – it leaves at least one superfluous double quote somewhere in the part. This effect can be negated by attaching it to a meaningless header, or as a meaningless attribute in an existing header. [27] also describes a "cross site file upload" which requires the Adobe Flex player.

But being able to predict the exact boundary string that will be used next, which is a by-product from the above research, enables a "clean" (i.e. identical to manually sent requests) construction of cross site file upload requests, requiring only HTML and Javascript (no Flash). The construction is per browser and O/S, and is possible (and tested) for:

- IE (Windows)
- Firefox (Windows, Mac OS/X, probably all BSDs)
- Safari (Windows)
- Chrome (Windows)

Note that conducting "cross site file uploads" can be used to effectively DDoS a website, e.g. if it scans each incoming file for viruses (which is very CPU intensive).

5.5 Detecting user behavior (IE only)

IE's boundary string exposes a unique piece of information – that of the foreground window handle. If an attacker manages to have his/her site rendered (even minimized) for an extended period of time by IE, the attacker can keep track of the foreground window handles. A-priori, the attacker cannot associate those handles to the actual applications/windows, but a careful analysis may still reveal some interesting data, such as when the user is idle, or when a user switches among many open applications vs. working in the same application for a long time.

It should be stressed that this technique leaks information on all applications – both browsers and non-browsers alike, i.e. it extends beyond the browser (IE) world and covers the whole desktop.