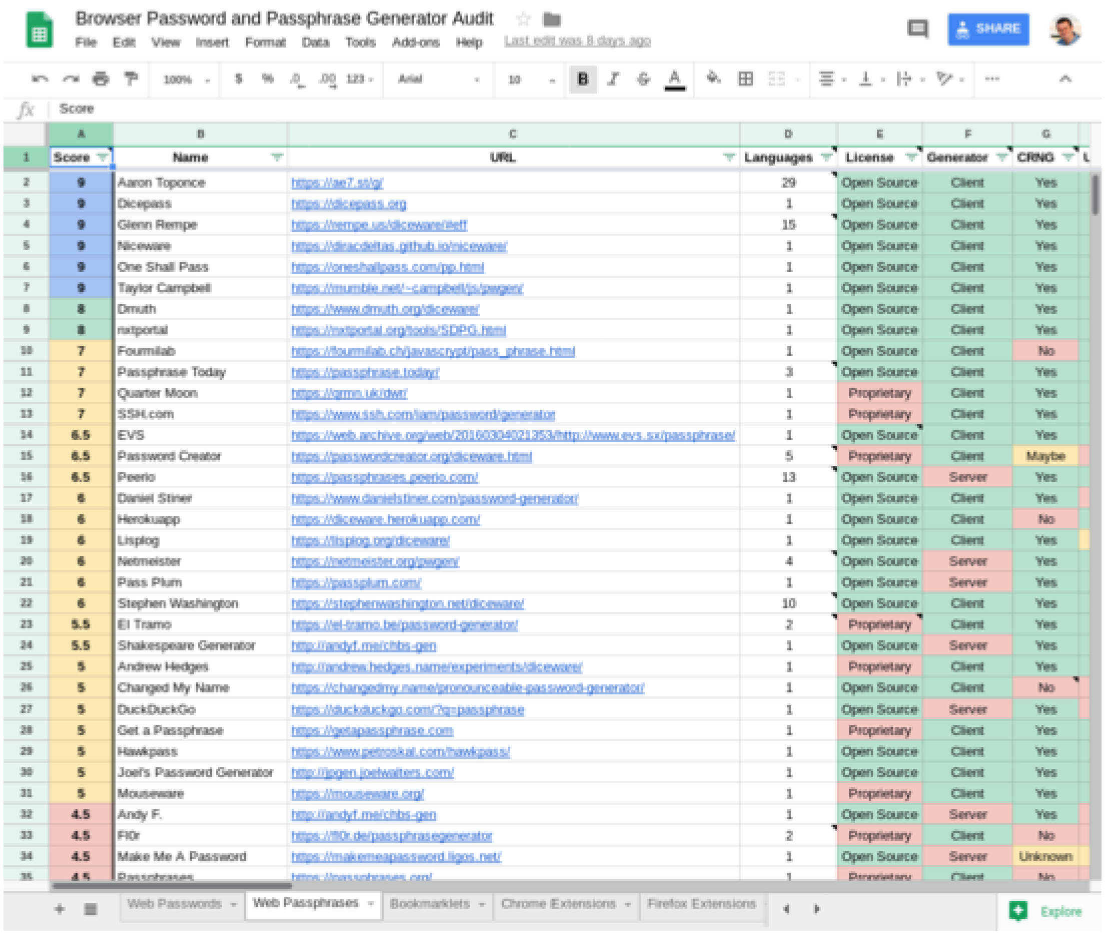
Aaron Toponce

{ 2018.04.19 }

Use A Good Password Generator

Introduction



Browser Password and Passphrase Generator Audit

Score	Name	URL	Languages	License	Generator	CRNG
9	Aaron Toponce	https://ae7.st/g/	29	Open Source	Client	Yes
9	Dicepass	https://dicepass.org	1	Open Source	Client	Yes
9	Glenn Rempe	https://rempe.us/diceware/#eff	15	Open Source	Client	Yes
9	Niceware	https://diracdeltas.github.io/niceware/	1	Open Source	Client	Yes
9	One Shall Pass	https://oneshallpass.com/pp.html	1	Open Source	Client	Yes
9	Taylor Campbell	https://mumble.net/~campbell/js/pwgen/	1	Open Source	Client	Yes
8	Dmuth	https://www.dmuth.org/diceware/	1	Open Source	Client	Yes
8	nxtportal	https://nxtportal.org/tools/SDPG.html	1	Open Source	Client	Yes
7	Fourmilab	https://fourmilab.ch/javascrypt/pass_phrase.html	1	Open Source	Client	No
7	Passphrase Today	https://passphrase.today/	3	Open Source	Client	Yes
7	Quarter Moon	https://qrmn.uk/dwr/	1	Proprietary	Client	Yes
7	SSH.com	https://www.ssh.com/iam/password/generator	1	Proprietary	Client	Yes
6.5	EVS	https://web.archive.org/web/20160304021353/http://www.evs.sx/passphrase/	1	Open Source	Client	Yes
6.5	Password Creator	https://passwordcreator.org/diceware.html	5	Proprietary	Client	Maybe
6.5	Peerio	https://passphrases.peerio.com/	13	Open Source	Server	Yes
6	Daniel Stiner	https://www.danielstiner.com/password-generator/	1	Open Source	Client	Yes
6	Herokuapp	https://diceware.herokuapp.com/	1	Open Source	Client	No
6	Lisplog	https://lisplog.org/diceware/	1	Open Source	Client	Yes
6	Netmeister	https://netmeister.org/pwgen/	4	Open Source	Server	Yes
6	Pass Plum	https://passplum.com/	1	Open Source	Server	Yes
6	Stephen Washington	https://stephenwashington.net/diceware/	10	Open Source	Client	Yes
5.5	El Tramo	https://el-tramo.be/password-generator/	2	Proprietary	Client	Yes
5.5	Shakespeare Generator	http://andyf.me/chbs-gen	1	Open Source	Server	Yes
5	Andrew Hedges	http://andrew.hedges.name/experiments/diceware/	1	Proprietary	Client	Yes
5	Changed My Name	https://changedmy.name/pronounceable-password-generator/	1	Open Source	Client	No
5	DuckDuckGo	https://duckduckgo.com/?q=passphrase	1	Open Source	Server	Yes
5	Get a Passphrase	https://getapassphrase.com	1	Proprietary	Client	Yes
5	Hawkpass	https://www.petroskal.com/hawkpass/	1	Open Source	Client	Yes
5	Joel's Password Generator	http://jpgen.joelwalters.com/	1	Open Source	Client	Yes
5	Mouseware	https://mouseware.org/	1	Proprietary	Client	Yes
4.5	Andy F.	http://andyf.me/chbs-gen	1	Open Source	Server	Yes
4.5	Fl0r	https://fl0r.de/passphrasegenerator	2	Proprietary	Client	No
4.5	Make Me A Password	https://makemeapassword.ligos.net/	1	Open Source	Server	Unknown
4.5	Passphrases	https://passphrases.org/	1	Proprietary	Client	No

Web Passwords | Web Passphrases | Bookmarklets | Chrome Extensions | Firefox Extensions

For the past several months now, I have been auditing password generators for the web browser in Google Sheets. It started by looking for creative ideas I could borrow or extend upon for my online password generator. Sure enough, I found some, such as using mouse movements as a source of entropy to flashy animations of rolling dice for a Diceware generator. Some were deterministic, using strong password hashing or key derivation functions, and some had very complex interfaces, allowing you to control everything from letter case to pronounceable passwords and unambiguity.

However, the deeper I got, the more I realized some were doing their generation securely and others weren't. I soon realized that I wanted to grade these online generators and sift out the good from the bad. So, I created a spreadsheet to keep track of what I was auditing, and it quickly grew from "online generators" to "password generators and passphrase generators", to "web password, web passphrase, bookmarklet, chrome extensions, and firefox extenions".

When all was said and done, I had audited 300 total generators that can be used with the browser. Some were great while some were just downright horrible. So, what did I audit, why did I choose that criteria, and how did the generators fall out?

I audited:

166.70.136.38 re license

- Server vs. client generation
- RNG security, bias, and entropy
- Generation type
- Network security
- Mobile support
- Ads or tracker scripts
- Subresource integrity

No doubt this is a "version 1.0" of the spreadsheet. I'm sure those in the security community will mock me for my choices of audit categories and scoring. However, I wanted to be informed of how each generator was generating the passwords, so when I made recommendations about using a password generator, I had confidence that I was making a good recommendation.

Use A Password Manager

Before I go any further, the most important advice with passwords, is to **use a password manager**. There are a number of reasons for this:

- They encourage unique passwords for each account.
- They encourage passwords with sufficient entropy to withstand offline clustered attacks.
- They allow storage of other types of data, such as SSNs or credit card numbers.
- Many provide online synchronization support across devices, either internally or via Dropbox and Google Drive.
- Many ship additional application support, such as browser extensions.
- **They ship password generators**.

So before you go any further, the Best Practice for passwords is "Use A Password Manager". As members of the security community, this is the advice we should be giving to our clients, whether they are family, friends, coworkers, or professional customers. But, if they are already using a password manager, and discussions arise about password generation, then this audit is to inform members of the security community which generators are "great", which generators are "good", which generators are "okay", and finally, which generators are "bad".

So to be clear, I don't expect mom, dad, and Aunt Josephine to read this spreadsheet, so they make informed decisions about which password generator to use. I do hope however that security researchers, cryptographers, auditors, security contractors, and other members of the security community to take advantage of it.

So with that said, let me explain the audit categories and scoring.

Software License

In an ethical software development community, there is value granted when software is licensed under a permissive "copyleft" license. Not necessarily GPL, but any permissive license, from the 2-clause BSD to the GPL, from the Creative Commons to unlicensed public domain software. When the software is licensed liberally, it allows developers to extend, modify, and share the code with the larger community. There are a number of different generators I found in my audit where this happened; some making great changes in the behavior of the generator, others not-so-much.

License	Open Source	Proprietary

So when a liberal license was explicitly specified, I awarded one point for being "Open Source" and no points for being "Proprietary" when a license either was either not explicitly specified or was licensed under a EULA or "All Rights Reserved".

This is something to note- just because the code is on a public source code repository, does not mean the
sed under a permissive license. United States copyright law states that unless explicitly stated,

all works fall under a proprietary "All Rights Reserved" copyright to the creator. So if you didn't specify a license in your Github repository, it got labeled as "Proprietary". It's unfortunate, because I think a lot of generators missed getting awarded that point for a simple oversight.

Server vs. Client Generation

Every generator should run in the browser client without any knowledge of the generation process by a different computer, even the web server. No one should have any knowledge whatsoever of what passwords were generated in the browser. Now, I recognize that this is a bit tricky. When you visit a password generator website such as my own, you are showing a level of trust that the JavaScript delivered to your browser is what you expect, and is not logging the generated passwords back to the server. Even with TLS, unless you're checking the source code on every page refresh and disconnecting your network, you just cannot guarantee that the web server did not deliver some sort of logging script.

Generator	Client	Server

With that said, you still should be able to check the source code on each page refresh, and check if it's being generated in the client or on the server. I awarded one point of being a "Client" generator and no points for being a "Server" generator. Interestingly enough, I thought I would just deal with this for the website generators, and wouldn't have to worry about this with bookmarklets or browser extensions. But I was wrong. I'll expand on this more in the "Network Security" category, but suffice it to say, this is still a problem.

Generation Type

I think deterministic password generators are fundamentally flawed. Fatally so, even. Tony Arcieri wrote a beautiful blog post on this matter, and it should be internalized across the security community. The "four fatal flaws" of deterministic password generators are:

1. Deterministic password generators cannot accommodate varying password policies without keeping state.
2. Deterministic password generators cannot handle revocation of exposed passwords without keeping state.
3. Deterministic password managers can’t store existing secrets.
4. Exposure of the master password alone exposes all of your site passwords.

Number 4 in that list is the most fatal. We all know the advice that accounts should have unrelated randomized unique passwords. When one account is compromised, the exposed password does not compromise any of the other accounts. This is not the case with deterministic password generators. Every account that uses a password from a deterministic generator shares a common thread via the master secret. When that master secret is exposed, all online accounts remain fatally vulnerable to compromise.

Proponents of deterministic generators will argue that discovery of the master password of an encrypted password manager database will also expose every online account to compromise. They're not wrong, but let's consider the attack scenarios. In the password manager scenario, a first compromise needs to happen in order to get access to the encrypted database file. Then a second compromise needs to happen in discovering the master password. But with deterministic generators, only one compromise needs to take place- that is using dictionary or brute force attacks to discover the master password that led to password for the online account.

With password managers, two compromises are necessary. With determenistic generators, only one compromise is necessary. As such, the generator was awardeded a point for being "Random" and no points for being "Deterministic".

Generator	Random	Unknown	Deterministic

166.70.136.38

RNG Security

Getting random number generation is one of those least understood concepts in software development, but ironically, developers seem to think they have a firm grasp of the basic principles. When generating passwords, never at any point should a developer choose anything but a cryptographic random number generator. I awarded one point for using a CRNG, and no points otherwise. In some cases, the generation is done on the server, so I don't know or can't verify its security, and in some cases, the code is so difficult to analyze, that I cannot determine its security that way either.

CRNG	Yes	Maybe	Unknown	No

In JavaScript, using a CRNG primarily means using the Web Crypto API via "window.crypto.genRandomValues()", or "window.msCrypto.getRandomValues()" for Microsoft-based browsers. Never should I see "Math.random()". Even though it may be cryptographically secure in Opera, it likely isn't in any of the other browsers. Some developrs shipped the Stanford JavaScript Cryptographic Library. Others shipped a JavaScript implementation of ISAAC, and others yet shipped some AES-based CSPRNG. While these are "fine", you really should consider ditching those userspace scripts in favor of just calling "window.crypto.getRandomValues()". It's less software the user has to download, and as a developer, you are less likely to introduce a vulnerability.

Also, RC4 is not a CSPRNG, neither is ChaCha20, SHA-256, or any other hash function, stream cipher, or block cipher. So if you were using some JavaScript library that is using a vanilla hashing function, stream cipher, or block cipher as your RNG, then I did not consider it as secure generation. The reason being, is that even though ChaCha20 or SHA-256 may be prediction resistant, it is not backtracking resistant. To be a CRNG, the generator must be both prediction and backtracking resistant.

However, in deterministic password generators that are based on a master password, the "RNG" (using this term loosely here) should be a dedicated password hashing function or password-based key derivation function with an appropriate cost. This really means using only:

- sha256crypt or sha512crypt with at least 5,000 rounds.
- PBKDF2 with at least 1,000 rounds.
- bcrypt with a cost of at least 5.
- scrypt with a cost of at least 16 MiB of RAM and at least 0.5s execution.
- Argon2 with sufficient cost of RAM and execution time.

Anything else, like hashing the master password or mouse entropy with MD5, SHA-1, SHA-2, or even SHA-3 will not suffice. The goal of those dedicated password generators or key derivation functions is to slow down an offline attempt at discovering the password. Likely the master password does not contain sufficient entropy, so it remains the weakest link in the generation process. By using a dedicated password hashing or key derivation function with an appropriate cost, we can guarantee a certain "speed limit" with offline clustered password attacks, making it difficult to reconstruct the password.

RNG Uniformity

Even though the developer may have chosen a CRNG, they may not be using the generator uniformly. This is probably the most difficult aspect of random number generation to grasp. It seems harmless enough to call "Math.floor(Math.random() * length)" or "window.crypto.getRandomValues(new UInt32Array(1))[0] % length". In both cases, unless "length" is a power of 2, the generator is biased. I awarded one point for being an unbiased generator, and zero points otherwise.

Uniform	Yes	Maybe	Unknown	No

To do random number generation in an unbiased manner, you need to find how many times the length divides the period of the generator, and note the remainder. For example, if using "window.crypto.getRandomValues(new UInt32Array(1))", then the generator has a period of 32-bits. If your length is "7,776", is in the case of Diceware, then 7,776 divides 2^{32} 552,336 times with a remainder of 2,560. This means that 7,776 divides values 0 through 2^{32}-2,561 evenly. So if your random number is between the range of 2^{32}-2,560 through 2^{32}-1, the value needs to be tossed out, and a new number generated.

Oddly enough, you could use a an insecure CRNG, such as SHA-256, but truncate the digest to a certain length. While the generator is not secure, the generator in this case is unbiased. More often than not actually, deterministic generators seem to fall in this category, where a poor hashing function was chosen, but the digest was truncated.

RNG Entropy

I've blogged about this a number of times, so I won't repeat myself here. Search by blog for password entropy, and get caught up with the details. I awarded one point for generators with at least 70 bits of entropy, 0.5 points for 55 through 69 bits of entropy, and no points for entropy less than 55 bits.

Entropy	70	69	55	54

I will mention however that I chose the default value that was presented to me when generating the password. Worse, if I was forced to chose my length, and I could chose a password of one character, then I awarded it as such. When you're presenting password generators to people like Aunt Josephine, they'll do anything they can do get away with as little as possible. History has shown this is 6-8 characters in length. This shouldn't be possible. A few Nvidia GTX960 GPUs can crack every 8 character ASCII password hashed with SHA-1 in under a week. There is no reason why the password generator should not present minimum defaults that are outside the scope of practical hobbyist brute force searching.

So while that may be harsh, if you developed one of the generators in my audit, and you were dinged for this- I'm calling you out. Stop it.

Network Security

When delivering the software for the password generation, it's critical that the software is delivered over TLS. There should be no room for a man-in-the-middle to inject malicious code to discover what passwords your generating, send you a determined list passwords, or any other sort of compromise. This means, however, that I expect a green lock in the browser's address or status bars. The certificate should not be self-signed, it should not be expired, it should not be missing intermediate certificates, it should not be generated for a different CN, or any other certificate problems. Further, the site should be HTTPS *by default*.

HTTPS	Yes	Not Default	Expired	No

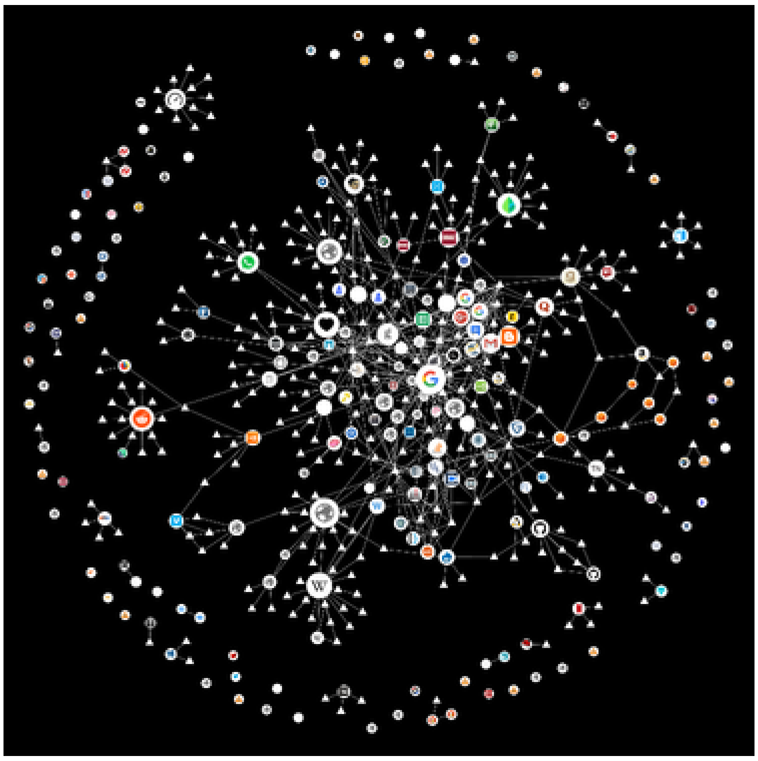
I awarded one point for "Yes", serving the software over secure and problem-free HTTPS, and zero points otherwise.

Mobile View Support

For the most part, due to their ubiquity, developers are designing websites that support mobile devices with smaller screens and touch interfaces. It's as simple as adding a viewport in the HTML header, and as complex as customized CSS and JavaScript rules for screen widths, user agents, and other tests. Ultimately, when I'm recommending a password generator to Aunt Josephine while she's using her mobile phone, she shouldn't have to pinch-zoom, scroll, and other nuisances when generating and copying the password. As silly as that may sound, if the site doesn't support a mobile interface, then it wasn't awarded a point.

Mobile	Yes	No

Ads and Tracker Scripts



I get it. I know that as a developer, you want easy access to analytics about who is visiting your site. Having that data can help you learn what is driving traffic to your site, and how you can make adjustments as necessary to improve that traffic. But when it comes to password generators, no one other than me and the server that sent the code to my browser, should know that I'm about to generate passwords. I awarded a point for not shipping trackers, and zero points if the generator did.

Trackers	No	Yes

Google Analytics, social media scripts, ads, and other 3rd party browser scripts track me via fingerprinting, cookies, and other methods to learn more about who I am and what I visited. If you want to see how extensive this is, install the Lightbeam extension for Firefox. This shows the capability of companies to share data and learn more about who you are and where you've been on the web. Recently, Cambridge Analytica, a small unknown company, was able to mine data on millions of Facebook users, and the data mining exposed just how easy it was for Facebook to track your web behavior even when not on the site.

At first, I thought this would be just something with website generators, but when I started auditing browser extensions, I quickly saw that developers were shipping Google Analytics, and other tracking scripts in the bundled extension as well.

Offline

When I started auditing the bookmarklets and extensions, I pulled up my developer console, and watched for any network activity when generating the passwords or using the software. To my dismay, some do "call home" by either wrapping the generator around an <iframe> or requiring an account, such as in the case with password managers. I awarded one point for staying completely offline, zero points for any sort of network activity.

Offline	Yes	No

Now, you may be thinking that this isn't exactly fair for password generators or just <iframe>s, but the generation is still happening client-side and without trackers. For the most part, I agree, except, when installing browser extensions, the developer has the opportunity to make the password generation fully offline. That may sound a touch silly for a *browser extension*, but regardless, it removes the risk of a web server administrator from modifying the delivered JavaScript on every page load. The only times the

JavaScript would be changed, is when the extension is updated. This behaves more like standard desktop software.

Subresource Integrity

Finally, the last category I audited was subresource integrity. SRI is this idea that a developer can add cryptographic integrity to <link> and <script< resources. The reason the W3C gives for the motivation of SRI is:

> "Compromise of a third-party service should not automatically mean compromise of every site which includes its scripts. Content authors will have a mechanism by which they can specify expectations for content they load, meaning for example that they could load a specific script, and not any script that happens to have a particular URL."

So SRI guarantees that even though the data is delivered under TLS, if the cryptographic hashes are valid, then the data has not been compromised, even if the server has. More information about SRI can be found at the W3C Github page, and I encourage everyone to check out the links there.

I awarded one point if "Yes" or "N/A" (no resources called), and zero points for "No".

SRI	Yes	N/A	No

Scoring

With all these audit categories taken into account, I gave a total score to see how generators ranked among others. Each generator has different auditing criteria, so the score varies from generator type to generator type. I treated the scoring like grade school- 100% is an "A", 99% to roughly 85% as "great", 84% to 51% is "okay", and 50% or less is failing. I translated that grade school score into the following:

Score	Perfect	Perfect - 1	Perfect - 2	51%	50%

When you look at the spreadsheet, you'll see that it is sorted first by score in descending order then alphabetically by "Name" in ascending order. It's sorted this way, so as a security consultant, you can quickly get a high-level overview of the "good" versus the "bad", and you should be able to find the generator you're looking for quickly. The names are just generic unique identifiers, and sometimes there are notes accompanying each generator when I found odd behavior, interesting generation templates, or other things that stood out.

Conclusion

This audit is for the security community, not for Aunt Josephine and friends or family. I don't expect you to share this with your dad or your spouse or your Facebook friends, and expect them to know what they're looking at, or why they should care. However, I recently took a poll on Twitter, asking the security community if they recommended password generators to family and friends. Only 90 votes came in, but 69% of you said that you did, with a few comments coming in saying that they would only recommend them if they shipped with a password manager (see the beginning of this post). I suspect that some of the 31% of the "No" votes are in this category.