

Posibles problemas en Qlink.it y librería CryptoJS

Antonio Castro Lechtaler¹, Marcelo Cipriano¹, Edith García¹, Pablo Lázaro², Julio Liporace¹, Eduardo Malvacio¹, Ariel Maiorano^{1,2}

¹ Grupo de Investigación en Criptografía y Seguridad Informática (GICSI),
Instituto Universitario del Ejército (IUE);

² Dirección de Gestión Tecnológica (DGT), Policía de Seguridad Aeroportuaria (PSA)
{acastro,marcelocipriano}@est.iue.edu.ar,
{editxgarcia,edumalvacio,jcliporace}@gmail.com,
{plazaro,amaiorano}@psa.gob.ar

Resumen. El objetivo del presente trabajo es describir los resultados preliminares obtenidos luego de una revisión del código fuente de la aplicación web Qlink.it. Este análisis inicial, aunque incluyendo aspectos relacionados a la seguridad de la aplicación de manera general, tuvo foco particularmente en lo relativo a su implementación de funcionalidades criptográficas. Si bien, salvo por la vulnerabilidad de XSS detectada, los potenciales problemas podrían no representar riesgos reales, el objetivo de esta publicación es también invitar a otros revisores a estudiar el sistema para confirmar si su utilización podría considerarse segura.

Palabras clave: seguridad informática, seguridad de aplicaciones, revisión de código fuente, criptografía, generación de números aleatorios, Qlink.it, CryptoJS.

1 Introducción

Recientemente se han publicado noticias [1,2] informando acerca de la disponibilidad del código fuente del sistema Qlink.it [3]. Considerando, por un lado, que el GICSI tiene por objetivo, entre otros, estudiar técnicas y mecanismos para la revisión de código fuente, con foco en los aspectos relacionados a la seguridad informática en general y a la criptografía en particular [4,5]; y por otro lado, que la DGT tiene por incumbencia, entre otras, evaluar periódicamente alternativas para la comunicación segura del personal de la Institución; se realizó conjuntamente una primera revisión general del código fuente de la aplicación web Qlink.it [6].

Al momento completar esta primera fase del análisis (mayo de 2017), los resultados preliminares de la revisión -aunque parcial e incompleta- advertirían posibles o potenciales problemas de seguridad, por lo cual se decidió consultar a los desarrolladores del sistema compartiendo estos resultados. Aunque se trate de una revisión que no abarcó a la totalidad del sistema, e inacabada, se solicitó el debido permiso para publicar esta información, en forma de artículo, con la intención de invitar a otros

revisores a estudiar el sistema, quienes podrían confirmar o rechazar estos potenciales problemas, y determinar si la utilización del sistema podría considerarse segura.

Es en este sentido también que un resumen limitado y sin mayores detalles de los primeros resultados fue publicado en el sitio dedicado a la seguridad de la información Segu-Info [17].

Se debe aclarar que uno de los potenciales problemas de seguridad tratados es “heredado” por la aplicación Qlink.it, al utilizar la librería CryptoJS [18].

Se aclara por último que algunos de los posibles problemas descriptos podrían no representar riesgo alguno, sin embargo se incluyen todos los resultados obtenidos para eventuales futuras revisiones por terceros y para a la vez dar cuenta del limitado alcance de la presente revisión.

1.1 Acerca de Qlink.it

De acuerdo a lo indicado en la documentación del proyecto [3,6], específicamente en su sección de preguntas frecuentes, “Qlink.it es una nueva manera, muy simple y segura, de enviar información confidencial a través de internet”.

Se detalla además que “Un qlink es un enlace normal de internet, pero con la característica especial de que se auto-destruye luego de que es leído por primera vez, aún si lo hubiese leído un robot. Como verás, esta característica especial es la pieza clave para enviar información confidencial a través de internet: si pones la información en un qlink, y envías el qlink a través de un correo o mensaje normal, entonces el destinatario sabrá que nadie más leyó la información si ésta estaba allí al pinchar el qlink.”

Modo de funcionamiento. Resumidamente, de acuerdo a lo informado también desde el sitio Web Qlink.it, en su sección de preguntas frecuentes avanzadas, el funcionamiento del sistema está descrito de la siguiente manera (en inglés en el original):

1. Cuando se ingresa un mensaje en qlink.it y se hace clic en el botón “qlink it!”, el navegador ejecuta código de Javascript que cifra el mensaje con una clave aleatoria dada, por ejemplo, YYYYYY.
2. Posteriormente, el mensaje cifrado se envía a través del protocolo https seguro al servidor Qlink.it.
3. En el servidor, el mensaje (ya cifrado con clave YYYYYY) se cifra de nuevo para ser almacenado, pero ahora con otra clave aleatoria, por ejemplo, XXXXXX.
4. Entonces, el servidor le devuelve un qlink preliminar, en este caso `https://qlink.it/XXXXXX`.
5. En ese momento, el navegador agrega al final del qlink preliminar la clave que sólo el navegador conoce para formar el qlink completo: `https://qlink.it/XXXXXX#YYYYYY`. Téngase en cuenta que el servidor Qlink.it no tiene acceso a la parte YYYYYY del qlink!
6. A continuación, es posible copiar y pegar el qlink completo y enviarlo al destinatario, ya sea por correo electrónico, chat, WhatsApp, o lo que sea.
7. Cuando el destinatario reciba el qlink completo y haga clic en él, el navegador sólo solicitará al servidor el qlink preliminar, `https://qlink.it/XXXXXX`,

porque la marca del carácter especial (#) indica que lo siguiente no debería ser enviado a través de Internet (Se puede comprobar esta función utilizando, por ejemplo, la opción de inspección en algunos navegadores como podría ser Chrome). Por lo tanto, el servidor Qlink.it nunca tiene acceso a la llave completa para leer el verdadero contenido del mensaje.

8. Cuando el servidor recibe la petición con el qlink preliminar, el qlink tiene en él la clave para buscar el mensaje encriptado y descifrarlo parcialmente. El servidor envía de nuevo a través del protocolo seguro https un mensaje que todavía está cifrado con la clave desconocida para el servidor YYYYYYY. En ese momento el servidor hace una eliminación segura en el mensaje encriptado y ya no está disponible en el servidor.
9. Cuando el navegador del destinatario obtiene el mensaje cifrado, ya que mantuvo la última parte del qlink completo YYYYYYY, ejecuta código Javascript para finalmente descifrar el mensaje usando esta última parte del qlink completo. Una vez que el mensaje se descifra totalmente, el navegador lo muestra en la pantalla del destinatario.

2 Posibles problemas en Qlink.it

Los siguientes potenciales problemas han sido detectados en una instancia del sistema instalada y configurada de acuerdo a las indicaciones provistas en [6]. Es por esto que los scripts que fueron desarrollados para las pruebas se encuentran configurados para trabajar con el servidor en “http://qlink/”. No se han realizado pruebas de ningún tipo en el servidor disponible en Internet, actualmente accesible en “https://qlink.it”.

2.1 Vulnerabilidad del tipo Cross-Site-Scripting (XSS)

Aunque se utilizan funciones Javascript para el filtrado de campos de entrada a completar por el usuario al generar un qlink –por ejemplo `filterXSS()` y `escapeHtmlEntities()`–, para el campo que contiene el mensaje, la entrada parece no verificarse o “filtrarse” correctamente. La salida del script de prueba anexada, “punto_2_1.py”, muestra que se simula lo que realizaría un navegador en cuanto a cómo es enviada la información al web-service que responde a estas peticiones en el servidor, especificando como mensaje, luego de un *tag* de cierre del elemento *textarea* que presenta el mensaje descifrado, un elemento `<script>` conteniendo una invocación a la función `alert()` para demostrar la vulnerabilidad frente a ataques del tipo XSS.

2.2 Otros parámetros de entrada y operaciones criptográficas

Luego de revisado el código en los archivos `public/js/application.js`, `app/src/Qlink/Models/Utils/RandomHasher.php` y `/app/src/Qlink/Controllers/LandingNewController.php`, se advierte que la primera parte del qlink, esto es, los primeros 10 caracteres, por ejemplo: `http://qlink/two/XXXXXXXXXXXX...` Son generados (no exclusivamente) en

base a un *timestamp* al milisegundo -resultado de la función Javascript `Date().getTime()` - que el navegador envía al servidor, y por lo tanto, que podría manipularse. Aunque el servidor registrará ese valor para usarlo en el próximo qlink, usando para la petición en curso el valor inmediato anterior, registrado previamente de la misma forma, el valor luego es sumado al *timestamp* del servidor, a la cantidad de microsegundos multiplicada por cien mil, y usado como semilla -mediante la función `mt_srand()` - para luego obtener valores a partir de la función `mt_rand()`.

Estas últimas funciones, basadas en el generador conocido como *Mersenne Twister*, no son aptas para utilizarse cuando existe la necesidad de números aleatorios para operaciones criptográficas, precaución explicitada también en su documentación oficial [7].

En el mismo sentido que lo anterior, ahora hablando del código que se ejecutará en el navegador mediante Javascript, aunque indirectamente, a través de la librería `CryptoJS` y su función `CryptoJS.lib.WordArray.random()`, la obtención de números aleatorios termina invocando a la función Javascript `Math.random()`, que es implementada por los navegadores en base a variantes del generador *Xorshift128+*, que tampoco es considerada segura o recomendable para la implementación de criptografía [8,9,10,11].

2.2.1. Estimación de fecha y hora de creación del qlink anterior

Sólo como un ejemplo que se desprende de lo anterior, se anexa también una salida de ejemplo de otro script de prueba, “prueba_2_2_1.py”, que requiriendo la generación de un qlink, tomará sus primeros diez caracteres para calcular la semilla utilizada, y realizar una estimación de cuándo fue creado el qlink inmediato anterior registrado. El script se vale del módulo o paquete `php_rand` [12].

Supóngase a x como el *unix epoch timestamp* del momento en que se generó el qlink anterior, en segundos; y a x_m al parámetro que se envió al servidor en tal momento, en milisegundos, por lo que a efectos de esta estimación aproximada, $1000x < x_m < 1000x + 999$. Por otro lado, asúmase t igual al *unix epoch timestamp* del momento en generamos el nuevo qlink, en segundos (suponiendo también que los relojes del cliente y servidor se encuentran sincronizados al segundo). Por último considérese a u como la cantidad en microsegundos utilizada en el código PHP de qlink, que de acuerdo a la implementación y recorte que sucede luego por número entero, se generaría tal que $0 < u < 99999$.

Por lo tanto, la semilla para el qlink que estamos generando, s , correspondería a $x_m + t + u$. Entonces, $s = x_m + t + u$, $s = 1000x + y + t + z$, con $0 < y < 999$ y $0 < z < 99999$. Luego, $x = (s - y - t - z)/1000$. Siendo esto una aproximación, se elimina el término $y/1000$, y $z/1000$ se reemplaza por un delta d , con $0 < d < 99$, entonces $x = [(s - t)/1000 - 99, (s - t)/1000]$. Por lo que la aproximación resultante corresponde a un rango de 99 segundos.

2.2.2. Obtención de "número DN" a partir de un qlink

También en base a los scripts de los ejemplos anteriores, se anexa la salida de “punto_2_2_2.py”, que muestra cómo sería posible obtener el código de seguimiento, o “número DN” de acuerdo a como es denominado en el sistema Qlink.it, a partir de los primeros diez caracteres de la parte variable de un qlink. Al arrancar, se genera un qlink para usar como ejemplo, pero el script podría adaptarse para funcionar con cualquier qlink que se encuentre vigente. El “número DN” es generado de manera similar a la primera parte de un qlink, sólo que el resultado corresponde a diez dígitos.

Se genera mediante una función muy similar a la anterior, radicando la diferencia en el conjunto de caracteres posibles para el mapeo de los números aleatorios. Se utiliza además el mismo *timestamp* que para la generación de la primer parte de un qlink. La función es invocada después de poco más de 50 líneas de código respecto a la generación de la primer parte del qlink. Por lo tanto, el script realizará pruebas para estimar el tiempo transcurrido entre la invocación a esas funciones para luego comprobar la existencia del código de seguimiento acotadamente, con la intención de reducir la cantidad de pruebas a realizar. Téngase en cuenta que puede ser necesario ejecutar varias veces el script para obtener resultados exitosos.

2.2.3. Generación de números aleatorios potencialmente insegura (Javascript, utilizando librería CryptoJS)

Volviendo a cuestiones relativas al código ejecutado en el navegador, específicamente en lo relacionado a la generación de números aleatorios, se anexa la salida de una prueba ejecutando “punto_2_2_3.py”, para demostrar potenciales problemas con la utilización de librerías basadas en la función que proveen los navegadores, implementando el generador *Xorshift128+*, de acuerdo a lo que fuera adelantado más arriba.

En este caso el script de prueba se trata de una adaptación de otro disponible en [14], que trabaja directamente con salidas de la función `Math.Random()`, valiéndose de la herramienta Z3, “*a high-performance theorem prover being developed at Microsoft Research*” [15], para la resolución simbólica del sistema de ecuaciones dada la información parcial conocida. El ejemplo de prueba fue adaptado para resolución con valores truncados por `CryptoJS.lib.WordArray.random()`. Se tomó a su vez como ejemplo la manera de generar el *salt* y el *iv* en qlink para estimar o adivinar valores posibles siguientes del generador. Si bien este ejemplo no representa de por sí un riesgo, debe considerarse que la misma función es utilizada para generar material de llave.

3 Posibles soluciones

Para los problemas descriptos en el apartado anterior, se proponen en lo que sigue, soluciones posibles para remediarlos o reducir el riesgo para los casos en que la amenaza fuera potencial. Adviértase de que de ningún modo deben considerarse a estas soluciones como correctas, completas o pertinentes. Se intenta únicamente comentar

resumidamente acerca de mecanismos de seguridad para sistemas, de manera general y en lo relativo a implementación de criptografía. Se sugiere a la vez la consulta de la referencia obligada en la materia, *The Open Web Application Security Project* (OWASP) [16].

3.1 Vulnerabilidad Cross-Site-Scripting (XSS)

Este puede tratarse de un caso a resolver especialmente ya que el servidor recibe un mensaje cifrado mediante el algoritmo AES. En principio entonces por supuesto no se podría verificar ni filtrar el contenido desde los scripts PHP que procesan, en el servidor, la información enviada. Por lo tanto la única alternativa sería aplicar filtros antes de presentar el contenido descifrado. Esto por supuesto mediante código Javascript a ejecutarse en el navegador del usuario. Revisando el código fuente del archivo `public/js/application.js`, se advierte que ya se estaría aplicando la función `JavaScript filterXSS()`, entre otras, sin embargo se comprueba que no funcionaría como se espera. Una posible solución, al margen de confirmar la correcta invocación de la función utilizada actualmente, sería reemplazar ese mecanismo por otro que nunca permita la ejecución de código Javascript al presentar el contenido en la página de lectura del mensaje descifrado.

3.2 Otros parámetros de entrada y operaciones criptográficas

A la hora de generar números aleatorios, se sugiere no utilizar funciones no seguras ni utilizar *timestamps* como semillas. También se desaconseja utilizar (directamente) otra información como códigos de teclas presionadas o posición del puntero. Frente a la necesidad de números aleatorios seguros, en Javascript, se recomendaría utilizar la función o método `RandomSource.getRandomValues()` de la nueva Web Crypto API, ya soportada en la mayoría de las versiones recientes de los navegadores más populares. Para el reemplazo de `mt_srand()/mt_rand()` del lenguaje PHP, la sugerencia sería obtener números aleatorios leyéndolos desde el archivo especial `/dev/urandom` (Linux). Alternativamente a ésta última recomendación, podría considerarse también la función para generar números aleatorios provista por el framework elegido por qlink, que se vale de la librería OpenSSL (utilizándose ya en el sistema para la generación de `x_tokens`). Se sugiere por último asegurar el manejo de números aleatorios y/o material de llave, para no recortarlos o truncarlos indebidamente en procesos de formato o adecuación al medio o presentación.

Como consideraciones generales, destacaríamos la conveniencia de realizar la autenticación, además del cifrado, del mensaje; por el navegador, al igual que el cifrado, sin compartir la llave con el servidor. Por ejemplo mediante un código de autenticación de mensajes, como podría ser HMAC, asegurando de esa manera la imposibilidad de modificación o alteración de esa información. Por otro lado, aunque se mencionó anteriormente, el control de abuso o prueba por fuerza bruta limitado al registro de la dirección IP demorando un segundo a cada petición, podría no ser la mejor alternativa disponible; quizá otro mecanismo criptográfico entre cliente (del web-

service) y servidor podría evitar otros abusos o pruebas de fuerza bruta aún potencialmente posibles.

4 Conclusiones

Los ejemplos del punto 2.2 pueden no representar problemas de seguridad concretos, pero debe destacarse la importancia de la posibilidad de manipular algunos parámetros utilizados como parte de semillas. Al margen de que el sistema tiene protección contra fuerza bruta limitando las peticiones a una por segundo por dirección IP, debería confirmarse si podría estimarse la fecha y hora del servidor a la manera en que está descrito por ejemplo en [13], y así, entre otras posibilidades, generar un mismo qlink repetidamente. Respecto al punto 2.2.3, convendría confirmar si el servidor podría o no conocer el estado del generador de acuerdo a la información recibida para estimar parte del material de llave del qlink.

Si bien, salvo por el primero de los casos, detallado en el punto 2, las potenciales vulnerabilidades podrían no representar un riesgo real, la utilización de herramientas y mecanismos de desarrollo diferentes a las recomendadas en buenas prácticas en relación a la seguridad en general, y a la implementación de criptografía en particular, hacen que se concluya prudente esperar otras revisiones del sistema o información ampliada por parte de sus desarrolladores.

5 Agradecimientos

Se desea agradecer a los desarrolladores y responsables del sitio Qlink.it por su rápida respuesta a nuestras consultas y su permiso para la publicación de los resultados preliminares. A su vez, se agradece también a Cristian Borghello por su ayuda en la publicación inicial resumida [17] en su sitio dedicado a la seguridad de la Información, Segu-Info, <http://www.segu-info.com.ar>.

Referencias

1. El físico argentino que creó un sistema de seguridad para e-mails. REVISTA NOTICIAS. [En línea] <http://noticias.perfil.com/2017/04/09/el-fisico-argentino-que-creo-un-sistema-de-seguridad-para-e-mails/>, accedido por últ. vez en mayo de 2017.
2. El acceso a mensajes encriptados por agentes de inteligencia vuelve al foco de debate. AGENCIA TÉLAM. [En línea] <http://www.telam.com.ar/notas/201703/183809-el-acceso-a-mensajes-encriptados-por-agentes-de-inteligencia-vuelve-al-foco-de-debate.html>, accedido por últ. vez en mayo de 2017.
3. Repositorio de Qlink.it en Github. [En línea] <https://github.com/qlinkit>, accedido por últ. vez en mayo de 2017.
4. Castro Lechtaler, A., Liporace, J., Cipriano, M., García, E., Maiorano, A., Malvacio, E., Tapia, N. Automated Analysis of Source Code Patches using Machine Learning Algorithms. XXI Congreso Argentino de Ciencias de la Computación (Junín, 2015). ISBN: 978-987-3806-05-6. [En línea]

- http://sedici.unlp.edu.ar/bitstream/handle/10915/50585/Documento_completo.pdf-PDFA.pdf?sequence=1, accedido por últ. vez en mayo de 2017.
5. Proyecto AAP, Repositorio de GICSI en Github. [En línea] <https://github.com/gicsi/aap>, accedido por últ. vez en mayo de 2017.
 6. Proyecto webapp, Repositorio de Qlink.it en Github. [En línea] <https://github.com/qlinkit/webapp>, accedido por últ. vez en mayo de 2017.
 7. Función mt_rand, Manual de PHP. [En línea] <http://php.net/manual/es/function.mt-rand.php>, accedido por últ. vez en mayo de 2017.
 8. Referencia de Math.Random() en Mozilla Developer Neywork. [En línea] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/random, accedido por últ. vez en mayo de 2017.
 9. Implementación de función Random() en CryptoJS. [En línea] <https://github.com/jakubzapletal/crypto-js/blob/master/src/core.js>, accedido por últ. vez en mayo de 2017.
 10. Implementación del generador XorShift128+ en Mercurial de Mozilla. [En línea] <https://hg.mozilla.org/mozilla-central/file/tip/mfbt/XorShift128PlusRNG.h>, accedido por últ. vez en mayo de 2017.
 11. Implementación del generador XorShift128+ utilizada en Chrome, Repositorio en Github. [En línea] <https://github.com/v8/v8/blob/master/src/base/utls/random-number-generator.h>, accedido por últ. vez en mayo de 2017.
 12. Implementation of mt_rand and mt_srand functions for bruteforce and speed, Repositorio en Github del proyecto. [En línea] https://github.com/Gifts/pyphp_rand, accedido por últ. vez en mayo de 2017.
 13. Argyros, G., Kiayias, A.: I Forgot Your Password: Randomness Attacks Against PHP Applications. En 21st USENIX Security Symposium. [En línea] <https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/argyros>, accedido por últ. vez en mayo de 2017.
 14. Symbolic execution for the XorShift128+ algorithm, Repositorio en Github del proyecto. [En línea] <https://github.com/dougward/XorShift128Plus>, accedido por últ. vez en mayo de 2017.
 15. The Z3 Theorem Prover, Repositorio en Github de los proyectos. [En línea] <https://github.com/Z3Prover>, accedido por últ. vez en mayo de 2017.
 16. The Open Web Application Security Project (OWASP). [En línea] <https://www.owasp.org/>, accedido por últ. vez en mayo de 2017.
 17. Posibles vulnerabilidades en Qlink.it (análisis web). Segu-Info. [En línea] <http://blog.segu-info.com.ar/2017/05/posibles-vulnerabilidades-en-qlinkit.html>, accedido por últ. vez en mayo de 2017.
 18. CryptoJS. Google Code Archive. [En línea] <https://code.google.com/archive/p/crypto-js/>, accedido por últ. vez en mayo de 2017.

6 Anexo

6.1 Pruebas y ejemplos relativos a los potenciales problemas tratados

Se anexan, como ejemplos de cada caso, las salidas de los scripts de pruebas desarrollados para la prueba de los potenciales problemas descriptos en el punto 2. El código

fuelle de estos scripts no se incluye en este anexo por cuestiones de espacio pero quien estuviera interesado podríu solicitarlo a los autores.

```
$ python punto_2_1.py
mensaje del qlink: Mensaje de prueba
mensaje para alert(): Prueba XSS
qlink: http://qlink/two/L3rpir1tqRPV8DUzVSPPGu#123456
$
```

Listado 1. Salida de la ejecución del script de prueba relativo al punto 2.1.

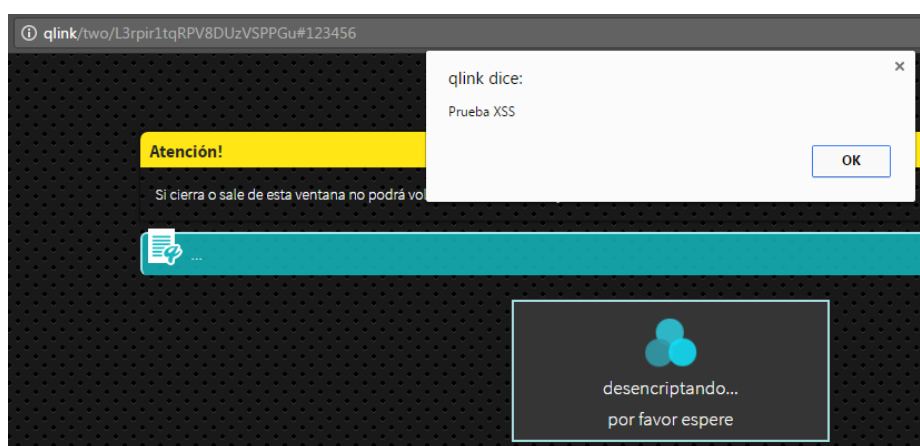


Figura 1. Captura de pantalla luego abrir el qlink generado.

```
$ python punto_2_2_1.py
qlink generado: http://qlink/two/f88cHwhwVDk14HN92XymdE#123456
obteniendo semilla...
semilla de qlink creado: f88cHwhwVD -> 1496390487982
el qlink anterior al recientemente creado fue generado, aproximada-
mente, entre Mon May 15 21:44:53 2017 y Mon May 15 21:46:32 2017
$
```

Listado 2. Salida de la ejecución del script de prueba relativo al punto 2.2.1.

```
$ python punto_2_2_2.py
qlink generado (ejemplo):
http://qlink/two/lQ9PvYWZrJGyTmqEvPeiiX#123456 dn obtenido:
8576041210
obteniendo semilla...
semilla de qlink generado: lQ9PvYWZrJ -> 1496555563119
```

```

qlink para timing #0 generado:
http://qlink/two/IoePfzTWYziTuniQBumReN#123456 dn: 4523295259
qlink para timing #1 generado:
http://qlink/two/lvaakCsFbIU8tBgSAuIqk#123456 dn: 7097969091
[...]
qlink para timing #9 generado:
http://qlink/two/cNR7bGgGzjdDPJ9hRN4N4t#123456 dn: 6566803103
diferencia entre semilla para qlink/dn #0: 1110
diferencia entre semilla para qlink/dn #1: 192
diferencia entre semilla para qlink/dn #2: 103
[...]
diferencia entre semilla para qlink/dn #8: 95
diferencia entre semilla para qlink/dn #9: 134
probando semilla: 1496555563214 trk: 3000169741 (0/1015)...
status: FAIL
probando semilla: 1496555563215 trk: 5334132004 (1/1015)...
status: FAIL
[...]
probando semilla: 1496555563652 trk: 7677823134 (438/1015)...
status: FAIL
probando semilla: 1496555563653 trk: 8097925954 (439/1015)...
status: OK
$

```

Listado 3. Salida de la ejecución del script de prueba relativo al punto 2.2.2.

```

$ python punto_2_2_3.py
se estimarán valores siguientes considerando salida conocida:
f7f195a46835dd93 f37e44f2afe372308cb09ca5c102323d
BROWSER: chrome
resolviendo...
modelo resuelto:
[c1480396913101804 = True,
 ostate0 = 14459375271365247648,
 ostate1 = 12664944705117373901,
 c149958108320120 = True,
 c1353346471225766 = True,
 c476870980001021 = True,
 c525365255680336 = True,
 c1015901882049230 = True]
siguientes 200 x 4bytes estimados: 7af3b8eac[...]
encontrados siguientes 16bytes generados en browser? True
$

```

Listado 4. Salida de la ejecución del script de prueba relativo al punto 2.2.3.