Pentest-Report Subrosa 05.2014

Cure53, Dr.-Ing. Mario Heiderich / Dr. Jonas Magazinius

Index

Intro

“Subrosa is an encrypted communication platform. Reclaim your privacy and talk freely. It's free and open source, with no ads.”

“With Subrosa, reclaim your privacy and dignity. All your communications are encrypted, and your encryption keys are never sent to Subrosa, and we can't read your messages or listen to your calls without your encryption keys, even with guns pointed.”

From https://subrosa.io/

This penetration test was carried out by two testers of the Cure53 team over the period of six days. The test identified six vulnerabilities of which five were classified as critical. Tests were carried out against the Subrosa application itself, its locally modified versions, the provided source-code and parts of the server-side code.

Scope

- **Subrosa Chat Application**
  - https://subrosa.io/app
  - Source-code provided by the Subrosa team

Test Chronicle

- 2014/05/16 - Penetration-Test begins
- 2014/05/17 - Standard XSS tests using malformed display name
- 2014/05/18 - Ongoing XSS tests, first Live-Reporting
- 2014/05/18 - Documented SR-01-001, SR-01-002, SR-01-004
- 2014/05/18 - WebSocket Debugging
- 2014/05/19 - Documented SR-01-003
- 2014/05/19 - Ongoing WebSocket Tests
- 2014/05/19 - JavaScript Source Code Audit
- 2014/05/20 - Analysis of login procedure
- 2014/05/20 - Analysis of crypto usage
- 2014/05/20 - Tested and verified XSS fixes in 0.16
- 2014/05/21 - Ongoing JavaScript Source Code Audit
- 2014/05/21 - Tests with rogue online status using *api.emit()* calls
- 2014/05/21 - Test for cross-site WebSocket hijacking
- 2014/05/21 - Analysis of WebSocket protocol
- 2014/05/21 - XSS Tests assuming server / TLS compromise
- 2014/05/21 - Added SR-01-005
- 2014/05/21 - Tested reliability of version check, added SR-01-006
- 2014/05/22 - Added SR-01-007, SR-01-008 and SR-01-009
- 2014/05/25 - Tested and verified XSS fixes in 0.17
- 2014/05/25 - Tested MitM capabilities for WebSockets
- 2014/05/25 - Tested against spoofed screen-sharing calls / phishing
- 2014/05/25 - Tests against extension of call duration / Info leakage
- 2014/05/26 - Security tests against website, FAQ and periphery
- 2014/05/26 - CSP compatibility tests, creation of SR-01-012
- 2014/05/26 - Tested array access security with users “constructor” and “__proto__”
- 2014/05/27 - Added SR-01-013
- 2014/05/29 - Finalization of Pentest-Report

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 for the purpose of facilitating future follow-up correspondence.

SR-01-001 XSS via unfiltered Display Name in Camera View Overlay *(High)*

There is an issue with the display name of a conversation partner in a video chat. Once it is shown in the camera overlay that displays the incoming video stream, it is not being escaped properly. This allows an attacker to inject arbitrary HTML and JavaScript whenever a video conversation between two or more participants is being initiated.

Steps to reproduce:

1. Log in with user A
2. Log in with user B
3. Log in with user C (malicious display name)
4. Have user A, B and C start a video conference
5. JavaScript executes

Affected Code:

app-webrtc.js #62ff

```
var id = "panel" + Math.round(Math.random()*999999999);
$("body").append("<div id='" + id + "' class='videoPanel'><video></video><div
   class='videoPanelActions'><div class='videoName'>" + (me ? "You" :
userdisplay)
   + "</div><div class='videoHangup'>End</div></div></div>");
var videoPanel = $("#" + id);
```

Result:

```
<div class="videoPanelActions"><div class="videoName"><iframe
onload="alert(1)">&lt;/div&gt;&lt;div
class='videoHangup'&gt;End&lt;/div&gt;&lt;/div&gt;&lt;/div&gt;</iframe></div></d
iv>
```

It is recommended to fix the vulnerability by using the *escapeText()* method on the *userdisplay* variable prior to echoing it:

```
$("body").append("<div id='" + id + "' class='videoPanel'><video></video><div
   class='videoPanelActions'><div class='videoName'>" + (me ? "You" :
escapeText(userdisplay))
   + "</div><div class='videoHangup'>End</div></div></div>");
var videoPanel = $("#" + id);
```

In the long run, it should be considered to use a JS MVC framework with auto-escaping to build the UI. A recommendation for a fitting and secure JSMVC framework is mentioned in the conclusion.

Note: Following our report, this issue was fixed in Release 0.16.

SR-01-002 XSS via unfiltered Display Name in Call Notification *(**Critical**)*

There is no proper escaping of the display name after it has been show in the call notification. An attacker can abuse this by first setting their own display name to be a malicious HTML string, and, secondly, calling the victim. Then and there, HTML and injected JavaScript will execute upon being rendered as part of the call notification.

Steps to reproduce:

1. Log in with user A
2. Login in with user B (malicious display name)
3. Have user B call user A
4. JavaScript executes

Affected Code:

app-view.js #1041ff

```
var icon = appcore.list[appcore.listHash[data.target]].avatar ||
"img/noavatar.png";
var name = appcore.list[appcore.listHash[data.target]].name ||
appcore.list[appcore.listHash[data.target]].displayname;
newNotification(icon, "Call from " + name, name + " is rining you..", 0, false);
startTitleAlert("CALL | Subrosa");
```

app-view.js #894ff

```
title = $("#htmlEscape").html(title).text();
content = $("#htmlEscape").html(content).text();
```

Result:

```
<div class="hide" id="htmlEscape"><svg onload="alert(4)"> is rining
you..</svg></div>
```

It is recommended to escape the value of the variable *name* using the global method *escapeText()*:

```
name = escapeText(name);
newNotification(icon, "Call from " + name, name + " is rining you..", 0, false);
```

Note: After we communicated this finding, the issue was fixed in Release 0.16.

SR-01-004 XSS via unfiltered Display Name in online Notification *(**Critical**)*

When pre-rendered for online-notifications, the display name is not being escaped properly. This gives an attacker the possibility to execute arbitrary JavaScript in the victim's browser by simply going online. An obligatory condition is that the attacker is a contact of the victim.

Steps to reproduce:

1. Log in with a user A
2. Make sure that a user B is a contact of the user A
3. Log in with the user B (malicious display name)
4. JavaScript executes

Affected Code:

app-view.js #1095ff

```
var icon = appcore.list[appcore.listHash["conv" + sortUID(data.uid,
appcore.uid)]].avatar || "img/noavatar.png";
newNotification(icon, data.displayname + " is online",
    statusText[data.newStatus], 5000, true);
```

app-view.js #894ff

```
title = $("#htmlEscape").html(title).text();
content = $("#htmlEscape").html(content).text();
```

The escaping method *escapeText()* should be used to mitigate this vulnerability as follows:

```
newNotification(icon, escapeText(data.displayname) + " is online",
    statusText[data.newStatus], 5000, true);
```

Note: Following our report, Release 0.16. benefits from this issue being successfully fixed.

SR-01-007 WebSocket Protocol vulnerable to Replay Attacks *(**Critical**)*

Subrosa claims to be resistant to Man-in-the-Middle attacks (MitM). While that may be true from a confidentiality standpoint, it does not hold from an integrity perspective. The communication protocol allows an attacker to replay a previously recorded communication. The "replayable" communication messages include, e.g., chat messages, and logins. An attacker who has a capacity to intercept an exchange of login messages between a client and the server can then replay these messages and effectively accomplish a successful authentication of the WebSocket session.

While the attacker cannot decrypt the users' information or messages, it allows for impersonation and grants an additional possibility for sending messages over a now-authenticated WebSocket session.

There are multiple consequences of this attack. One scenario is that the attacker replays further intercepted messages to, for example, spoof chat messages in the name of the victim-user.

Steps to reproduce chat message spoofing:

1. User B (the attacker) is performing a MitM attack between user A (the victim) and Subrosa.io
2. User A logs in
3. User B records the authentication message
   `["{\"step\":2,\"username\":\"internot\",\"hash\":\"b4f298d4afbd06a6888a2b19cc14fa39c8fd879f500aa44d29ebda39019fd841\",\"sockType\":\"loginMain\"}"]`
4. User A sends a chat message to C (either a user or a group chat)
5. User B records the encrypted chat message
   `["{\"target\":\"conv2ecSPdGMr8Qgf5ry-FT3wBSx9xlyQxrLi\", \"type\":2,\"data\":\".4\\u000e²Ó\\u0014Ø▫\\u001a3Ï\\u000e__▫Øw\",\"auxdata\":\"ü9úÒÖDPÃhó¶\\u000b'L\\u001f\\b\",\"sockType\":\"comm\"}"]`
6. User A logs out
7. User B replays the authentication message
8. User B replays the chat message
9. C receives the chat message that appears to originate from user A

A scenario of greater impact empowers the attacker enough for a complete hijacking of the victim's account. After using the replay attack to authenticate a WebSocket, the attacker follows the attack strategy detailed in SR-01-009 to replace given user's profile with a one encrypted with the attacker's password. Including a time-stamp in the encrypted message is an effective way of preventing replay attacks. Consequently, each encrypted message is unique and the time-stamp is easily verifiable on the receiving side.

SR-01-008 Manipulation of IV changes Decryption Output *(**Critical**)*

In AES CBC mode the first block is "XORed" with an initialization vector (IV) before encryption. This is to ensure that the encryption of two messages with identical first blocks produces the same cipher-text. Consequently, the output is again XORed with the IV as the last step of decryption. This gives the IV a direct influence over the decrypted plain-text. By manipulating the IV, an attacker can now arbitrarily alter the first block of the plain-text. The first block is limited to the initial 16 characters of the output, and the implications depend entirely on what information is contained in these characters.

Subrosa is impacted in a way that chat messages can be forged to a certain extent. In combination with SR-01-007 this allows an attacker to forge messages in the name of another user. In the example below, the second message is the encryption of the plain-text chat message "wow". By manipulating the IV as follows, the received message becomes "vov":

```
["{\"target\":\"conv2ecSPdGMr8Qgf5ry-FT3wBSx9xlyQxrLi\",\"type\":2,
   \"data\":\".4\\u000e²Ó\\u0014Ø▫\\u001a3Ï\\u000e_▫9w\",
   \"auxdata\":\"ü9úÒÖDPÃió¶\\u010b'L\\u001f\\b\",\"sockType\":\"comm\"}"]
```


Two components assist in mitigation of this issue, namely signing the messages and verifying the signatures upon receipt. This issue can be mitigated by signing the messages and verify the signatures upon receipt. Thereby a message with manipulated IV would be decrypted to a plain-text with an incorrect signature.

SR-01-009 Password update leads to full account compromise *(**Critical**)*

In a scenario where the attacker can successfully carry out a MitM attack, he or she can either inject messages into the open channel or, alternatively use the replay attack (described in SR-01-007) to fully compromise the account. By abusing the password update functionality the attacker can “change” the password. This is done when a sequence of messages (“updateBlob” and “changeProfile”) is sent with a completely new profile controlled by the attacker. As a result, the original (real) profile is replaced with a forged one encrypted with the attacker’s password. The existing profile is destroyed in the process, however, the victim’s contacts remain unaware of the switch and have no reason to mistrust the username. The attacker can abuse this confidence to engage in communication with the unknowingly tricked contacts. The attack works because the server sees the WebSocket as trusted and there is no server-side verification of the channel’s integrity in place.

To mitigate this issue it is recommended that the profile is encrypted with both the old and the newly-derived key, rather than relying on the newly created derived key exclusively. Ultimately, this should ensure that the user has knowledge of both the old and the new passwords.

Injected messages:
The following two messages are required to replace a profile’s password with the string “testtesttest”

1. The *updateBlob* message:


```
["{\"iv\":\"▫▫▫·è\\u0015Ð°QÕw▫
▫;l2\",\"blob\":\"rvZr4VCxEteloUgjuXR42pjSLrgMKZnmO7V3Wzz4VUtJ87zzGLun3CUK2B0AqQ
JddNC2cO9JMgtkRqzPs7BP/Jdg1rUV/PB9jHp9RknfSSzym4pMc3a8x+mw0Gvn/UDVJMYL+z3GhUTbqn
j42O9GfVNcDqmdGAf0RsWACnbCtKkv/39I/DzeA75AJkUyN4jXkdJPaV8ej2BK/gkucfkb/1YIHjuxFS
sDt+sPa0p0ZT5BcaQhPgIgUFtHFSzs/1w2/w4QaiUqoCtHiMURJSWMSIr/SwzGXzA+
+sTfZanFHRS8ZtLnHtfiUeBnKbua9iVGaW5D2hSR5Td0Lb8qJto8E6gOhu+sF3V36SBkJBMFcRcEkOk0
F1uDPOsGTgygq9rx8ZmEV8GhSavxpWPYBHc9rfkPWL6rMcuDiat7jR64/uqt42l/GezdITQUyGp3YeO+
xwzfRQFo0ufpTF9KW9N1uU7QIcvT5x/lKsgo0Sg6sEGQFWuidnQQHpVpZEzGNyinytj+OvpzDvIz21ZR
mxeS+T4f+2jHuABdDIPYprH/pW0aCTKkxD+A0v1fxmqXVCeMcpKxHTAC9jLDzlshNKOGRMxD3UIQI7qc
FYSebb...
wwCtEm5mboassg13oW33goIyoMyCLaAu+35G1oqvdODZTWuBuV2ECOALhB3JwyLkYirJWFCLiig+3CQJ
3aubYdd0UtI+Ed5Uyp4r84OI1spjVD90a4UBgZgg4Oe3Q9Rqe4HvUawgivY/3PlO+n2Drlp4KtJ8bDa+
91CtIIOS8Zb/ekkfnzimr6zerMq/R8uH1Wspbi2GeTxlryw1+CPkKxpdb9LP6gJRMo5APe9cwnRIdot5
f/C8M9ud726MJqYVrjZlOCB2nPO8xY+
+GIIcR+M1PocBjz3GQO1wNbY3ENQnqakt+Wjv79IgP/wjnIYw3ngGnS5WEkIg5CL5uTob79a8KIHIuuL
xEEqHrYScqPEOkk8BUmJLUO4+X+g2BPyapYL9VfylNUOG/ebT26V+U5qs2c8gO9te8C9KJUvZ0t290Oq
```

ZQkOcc+tbtvrBLXSxw2PywwPbTNI3kU5cqLvvACRsF6YKGkk8HF70DxZ+8ya/Z2R0wtaRkcnwMK9kaPo
njZ7fv5pMCq/ZOAkh146Jy1JpK20obAE23oUJ6lQj7zEtgqE1K4XxlQOicYNdXdE1TQZ8ysv0ZaJl2aS
nS9/x5qNkWlB90OvQ5t8a0xtlcsASWIE4mt+tE9LpkjZ5o3AGHtRqf202XcNxLS/NcCbnXf8vJrG7IU+
bJNfBGLJckmTZhLZDQKWF71M=\",\"sockType\":\"updateBlob\"}"]

2. The *changeProfile* message:

["{\"newpass\":\"\",\"oldpass\":\"\",\"derivedKeyHash\":\"b64097a49ad80c98cf4d37
9e201d427a103b993e5de14cb3e43a08357f282ecd\",\"derivedKeySalt\":\"Ô▫?__
êÊÑX`xÃÑ\\u0003▫Ü#▫lêú|Ó±OjÂ\\n¶\\u001cèÉÉ\",\"sockType\":\"changeProfile\"}"]

SR-01-013 Call Extension Attack allows for covert Surveillance *(High)*

It is possible for an attacker to make an ongoing call visually appear as though it has been terminated. This allows the attacker to covertly surveil any audio in the vicinity of the victim's computer. The browser will indicate that audio is still being recorded, but the call is not visible in the interface.

When a call has been established, the interface uses a green indicator bar to convey that the call is active and includes a button for hanging up. If the user receives a second call from another user and decides to take it, the preceding call will be ceased. However, if the second call is made by the same user, the indicator bar of the second call replaces the one of the first call, whereas the original call remains connected. When the second call is finished, the indicator bar disappears entirely, which not only makes the first call appear as 'hung up' but also leaves the user (call recipient) with no controls to actually terminate it. The call continues to be recorded until the caller decides to end it. By logging in with the same account twice and setting the microphone to 'mute', an attacker can use this vulnerability to covertly record audio from a victim's computer for an extended period of time.

Steps to reproduce:

1. The attacker logs in with his or hers account in one browser window or tab
2. The attacker makes an audio call to the victim
3. The victim answers the audio call
4. The attacker logs in with the same account in a separate window or tab
5. In the second window, the attacker makes a call and immediately hangs up
6. The attacker mutes the microphone
7. The first call now appears to have ended but, in fact, the connection remains open

The only giveaway of the call being still ongoing is the browser's recording indicator.

- For Chrome it takes shape of a small red recording icon shown on the tab hanger
- For Firefox it is in the form of a small green camera icon shown in the address bar

To mitigate this vulnerability, the client must always verify that an existing call from the same user is not already established.

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid an attacker 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.

SR-01-003 Possible passive XSS via unfiltered News Item URL *(Low)*

As the Subrosa news page receives news items via WebSocket and JSON, it displays the resulting data on the index page immediately after login takes place. Under the unlikely assumption that the Subrosa server is compromised, an attacker might be able to inject non-HTTP URLs and thereby affect the victim's security and cause XSS.

Example JSON Data:

```
a["{\"news\":[{\"title\":\"Project v0.1 Prebeta!\",\"content\":\"Project v0.1
prebeta finalized! All that's left is putting this up on a host and purchasing
domains!\",\"link\":\"https://example.org\",\"time\":1397043420794},
{\"title\":\"meow Sample Article\",\"content\":\"This is just another news
article for Project. It should just be a paragraph or so, as a small sample.
There should not be too much content
here.\",\"link\":\"https://example.org\",\"time\":1396914700210}],\"sockType\":\
"news\"}"]
```

Malicious JSON data:

```
a["{\"news\":[{\"title\":\"Click me, I am a dolphin\",\"content\":\"Project v0.1
prebeta finalized! All that's left is putting this up on a host and purchasing
domains!\",\"link\":\"javascript:alert(1)\",\"time\":1397043420794},
{\"title\":\"meow Sample Article\",\"content\":\"This is just another news
article for Project. It should just be a paragraph or so, as a small sample.
There should not be too much content
here.\",\"link\":\"https://example.org\",\"time\":1396914700210}],\"sockType\":\
"news\"}"]
```

Resulting HTML:

```
<div class="homeNewsElem"><a href="javascript:alert(1)" target="_blank"><span
class="title">Click me, I am a dolphin!</span></a><span class="time">Apr
9</span><span class="content">Project v0.1 prebeta finalized! All that's left is
putting this up on a host and purchasing domains!</span></div>
```

It should be ensured that URLs are either relative or cannot use schemes other than HTTP and HTTPS. While being compromised as a result of this weakness is highly unlikely, all necessary precautions should be taken to narrow the surface for user-compromising.

Note: Following our report, this issue was fixed in due course for the Release 0.17.

SR-01-005 Reliance on Server-Sanity causes tremendous XSS Risk *(Medium)*

In its current state, the security model Subrosa uses comes down to a strict reliance on the fact that the server delivering content via WebSocket protocol is not a victim of an attack. Once the server is in fact under attack, the entire security model falls and each and every user of Subrosa will be prone to XSS attacks. The very same pattern is valid for (hypothetical) successful attacks against SSL[1]. The reasons behind that lie in the Subrosa client's overly extended trust in the content sent by the WebSocket server and using the received data in complex, unescaped HTML concatenations before rendering them to the user.

An example of the described problem can be found in the file *app-view-uisetters.js* as seen below.

Affected Code:

app-view-ui-setters.js #282ff

```
function createItemHTML(type, item){
    if(type == "conv"){
        var pinnedIcon = '<span class="listItemPinned fa fa-thumb-tack"'
              + (getProp(item.id + "-pinned") ? ''
              : ' style="display: none"') + '></span>';
        if(item.id.indexOf("-") != -1){
               var subtitle = (typeof item.displayname
                     != 'undefined' ? item.username : "");
               if(!subtitle){
                      if(item.contact == 0){
                             subtitle = "Not contacts";
                      } else if(item.myRequest){
                             subtitle = "Sent request";
                      } else {
                             subtitle = 'Contact request';
                      }
               }
               return '<div class="sidebarListItem'
          + (currentTab==item.id ? ' activeItem' : '')
          + '" data-item="' + item.id
          + '" data-trigger="conv"><div class="unreadBadge">0</div><img src="'
          + (item.avatar ? item.avatar : 'img/noavatar.png')
          + '" class="listItemIcon" data-status="'
          + (typeof item.status != 'undefined' ? item.status : "")
          + '" /><span class="listItemTitle">'
          + escapeText(item.displayname || item.username)
          + '</span> ' + pinnedIcon + '<br /><span class="listItemSubtitle">'
          + subtitle  + '</span></div>';
           } else {

          ...
    }
}
```

[1] http://carbonwind.net/blog/post/A-quickie-for-a-Friday-e28093-a-SSLTLS-timeline.aspx

Once an attacker has control over the data emitted by the WebSocket server or can carry out a successful MitM attack, it is possible to inject arbitrary HTML and JavaScript and take control over the user's browser. This signifies getting access to contacts, fingerprints and clear-text messages. An example response that would cause immediate XSS hazard for the attacked users would look like this (and cause a new element to be added to the sidebar, infected via the *items.id* property):

```
a["{\"object\":
{\"contact\":0,\"myRequest\":false,\"username\":\"x00mario2\",\"bgColor\":\"F2E8
FF\",\"uid\":\"hrGuLjpF2xO5Fx3u\",\"id\":\"\\\"><img src=x
onerror=alert(1)>conv3ebDjU8ASf1kb1RG-
hrGuLjpF2xO5Fx3u\",\"lastMyMessage\":\"0\",\"lastMessage\":\"1400676288877\",\"l
astRead\":\"0\",\"users\":
[],\"usercount\":\"1\",\"created\":true},\"autojoin\":true,\"sockType\":\"newLis
t\"}"]
```

Note that this is not the only file where XSS becomes possible in case the server is compromised. This ticket is intended to point out the general risk and suggest mitigation strategies rather than simply list all related findings. In order to fulfil the crucial goal of keeping Subrosa's security promises, it is recommended to consider two approaches:

- Move away from creating template strings which employ HTML concatenation. This has been repeatedly deemed 'bad practice' and should not be used in any security-critical software. It is recommended to start making use of a templating system that filters data by default and thereby takes over the developer's burden of fixing XSS reactively.
- Do not invest any trust in data that is being returned from the server. The long history of SSL bugs to date and a shared expectation of there being more to come put the server in danger. Currently, the server is posing a threat of being a single point of failure. Each and any incoming bit of data should be considered untrusted and validated or escaped accordingly.

Note: Our report inspired a fix to this issue in Release 0.17.

SR-01-006 Subrosa Version Check can be bypassed *(Low)*

Whenever Subrosa starts, it checks whether a new release is available. If this is the case, the login button is being disabled. By re-enabling the button, a user can still log in and use the software in the not up to date version. As such, users who never log out might not notice a presence of an update and unknowingly stick to a vulnerable version.

It is recommended to have the application check for new releases regularly rather than conditioning the check upon the login form being displayed. This could be implemented via a heartbeat or similar mechanism to make sure that the time frame between exposure to attacks and a message urging an upgrade is as short as possible.

SR-01-010 User information partially sent in Cleartext *(Medium)*

While chat messages are encrypted, information such as the user names, user-IDs, and conversation-ids are communicated in plain-text. This does not disclose information on the content of a conversation, but vitally grants access to potentially sensitive information on who is communicating with whom. The severity of this issue depends on the assumed threat model and the expectations of potential customers. This is a design choice made by the developers, therefore classified as miscellaneous issue of medium severity impact.

SR-01-011 UI can be broken using localStorage.sidebarWidth *(Info)*

An attacker with short physical access to the victim's browser can change the values stored in Subrosa's *localStorage* array. This means that an attacker can set the sidebar width value to a negative value, thereby destroying the UI after the victim has logged in successfully. To avoid this, the value for the sidebar width should be validated to be a positive integer in a range from 100 to *screenWidth*-n.

SR-01-012 No X-Frame-Options Headers or any Form of CSP is used *(Low)*

In its current state, Subrosa does not make use of any HTTP Security headers. While Clickjacking has a very low risk factor for client-only apps, using CSP to avoid XSS and script injections[2] should be considered. As it stands, Subrosa makes use of JavaScript URIs but a quick analysis of the code for general compatibility with CSP yields a positive impression. The following CSP rule example could be used as a template for the initial experiments:

```
Content-Security-Policy:
       default-src 'self';
       img-src 'self' data:;
       connect-src wss://subrosa.io/ https://subrosa.io/;
       style-src 'self' 'unsafe-inline'
```

It needs to be kept in mind however, that the current way of handling form submits using event handlers and "return false" should be avoided to make the app compatible with the restrictions imposed by CSP. Using CSP essentially excludes the option of having Subrosa vulnerable to a majority of XSS attack vectors. Templating frameworks, such as AngularJS[3], are meanwhile compatible with CSP and allow developers to benefit from the security advantages of both worlds.

[2] http://www.w3.org/TR/CSP11/
[3] https://angularjs.org/

Conclusion

Subrosa aims to become a secure, privacy-aware and lean replacement for communication tools such as Skype. Running entirely in the client, it is driven by modern technologies like HTML5, WebSockets[4] and WebRTC[5]. As such, it allows users to chat, conference, make calls, group calls and even video calls, without an actual client, employing solely the browser. The Subrosa backend is designed in a way that almost no information on what the users are doing is known. The browser utilizes the Forge library to encrypt all sensitive data directly on the client and thereby maximizes the security and privacy aspects that Subrosa is offering. However, by choosing this path, Subrosa is confronted with security threats that normally affect classic web applications and browser extensions. This signifies vulnerability to XSS, DOMXSS, and HTTP leaks, as well as the presence of very easy ways for attackers to create rogue clients and send arbitrary data to the Subrosa servers, potentially even transmitting it to other connected clients.

The first tests yielded several severe XSS problems that allowed attackers to send arbitrary HTML and JavaScript to other users and thereby gain control over the DOM of their Subrosa clients. This means access to any information loaded in the Subrosa domain, inclusive of the entirety of stored conversations in plaintext, hijacking audio and video streams and other sensitive data. The identified and reported fixes were addressed very quickly and professionally by the Subrosa Team. Continuously, the current code-base can be considered XSS-free, as proven by thorough black-box tests as well as a source code analysis, which encompassed an examination of all sources and sinks for user-generated data.

Nevertheless, it should be considered to rebuild the Subrosa UI and make use of a secure-by-default template engine where each and every bit of user generated data is being escaped unless explicitly demanded otherwise. Furthermore, Subrosa does not yet make use of any CSP policy despite the act that, for applications running entirely in the browser, a strict policy is mandatory. Cure53 conceived a test-policy for Subrosa and managed to create a fairly well-operating state, making the pairing of Subrosa and CSP possible and highly recommended for future releases[6]. Ultimately, a rather unique threat model for Subrosa is the fact that presently an attacker can log in to multiple instances at the same time and perform almost unlimited amounts of actions, such as numerous calls to the same users using the same account and the like. This caused the call extension attack described in SR-01-013 to work and might yield additional issues if not handled properly.

[4] https://developer.mozilla.org/en/docs/WebSockets
[5] http://www.webrtc.org/
[6] SR-01-012 No X-Frame-Options Headers or any Form of CSP is used

Subrosa might still have a long refactoring-centered way to go, but is nevertheless a highly attractive alternative to Skype even in its young state. Subrosa's key strengths pertain to the capacity of providing features that support productivity while at the same time setting the security bar high. We believe that future versions will accomplish the goal of tackling the remaining concerns and result in putting forward a communication application that is ahead of its time, providing the level of security that the times we live in demand.

Cure53 would like to thank the Subrosa Team for their support and assistance during this assignment.