Security Analysis of Open Home Automation Bus System

Milan Ramljak
University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
ramljak.milan@gmail.com

Abstract - Today's modern homes are becoming complex live systems in which virtually all functionality, from lighting and heating control to security and occupancy simulation, is mediated by computerized controllers leading to IoT future. The smart nature of these homes raises obvious security concerns and history has shown that a vulnerability in only one component may provide the means to compromise the system as a whole. Thus, the addition of every new component, and especially new components with external networking capability, increases risks that must be carefully considered. In this paper we examine one of the most active open source home automation framework, Open Home Automation Bus (openHAB) which is used as platform for many other IoT supported devices. First, we go through openHAB security architecture and supported features following the challenge of a static source code analysis of several most used openHAB packages (called bindings) and carefully crafted test cases that revealed many undocumented features of the platform. Next, we exploited security flaws by constructing two proof-of-concept attacks that: (1) openHAB system denial of service; (2) inject custom binding for message bus monitoring and control; We conclude the paper with security best practices for the design of custom openHAB bindings.

I. INTRODUCTION

With the unstoppable advance of technology, most research projects in field of smart homes have been designed to help human and increase their quality of everyday living experience. A home, which is smart, is the technology result used to make all electronic equipment around the home act "intelligent" or more smart. Smart home has advanced automatic rules for lighting, temperature control, security and many other functions. Each and every new home appliance are having embedded intelligence with automation, monitoring and controlling possibilities. Exposing the data outside smart home system, except security concerns, provides wide range of potential benefits for users and companies building those systems. There are many smart home solution, but of interest in this paper is Open Home Automation Bus (openHAB) system with main focus on its security aspects. We chose openHAB for several reasons. First, openHAB has a growing community driven approach. Second, connection to 153 different devices and protocols such as Samsung TV, Z-Wave, Asterisk and many other. And third, source code is available giving easy way for analysis.

OpenHAB is a software for integrating different home automation systems and technologies into one single solution that allows reusable automation rules and offers uniform user experience. OpenHAB does not try to replace existing solutions, but rather wants to enhance them - it can thus be considered as a system of systems. A core concept for openHAB is the notion of an “item”. An item is a data-centric functional atomic building block. [1] Under the hood, openHAB does not directly communicate with a physical device. Instead, it communicates with an instance of an “item” that encapsulates a physical device. This approach makes it even easier to replace device communication protocol by another without doing any changes to rules and UIs.

A most advanced aspect of openHAB’s architecture is its modular design giving simple way to extend system at runtime without stopping it. Each feature is developed as completely standalone Java OSGi library also known as openHAB binding. Modular binding approach has been a huge enabler for the active community around openHAB with many engaged contributors [2]. Looking from security side this could lead to many problems in security implementation as different bindings are developed by different users implying that static code analysis is important. Our security analysis explores the above security-oriented aspects of the openHAB framework. Performing the security analysis of an open-source system revealed devastating result of binding documentation. Some bindings are well document but some without any documentation at all, referring to code documentation. For an experienced Java developer is not that hard to dig into the code to get more understanding but some more complex bindings do need significant amount of time. To overcome these challenges, we used a combination of existing static analysis tools and manual analysis on a dataset of 153 bindings that we downloaded in source code form. As openHAB is based on Java programming language for this purpose we used Java enabled analysis tool Find Security Bugs.

II. RELATED WORK

Although the openHAB system analyzed in this article is widely used within the open source community, to day of writing this article, no information of existing security assessments has been found. Still many researchers touched the field of smart home systems and their security implications. Except the analyzed system, there are few other open source solutions such as openHAB [12]:

- Domoticz – a home automation system with native HTML5 support, developed in C/C++ programming language
- OpenMotics – open source system designed to deliver full wired solution, including software and hardware
- Calaos – full-stack home automation platform, including preconfigured Linux operating system

Search for any structured security analysis documents related to any of above mentioned resulted without success. Most of the security analysis articles focus only to smart home devices and their protocols, building blocks, and barely on the complete system utilizing them. All automation systems support set of widely used smart devices such as switches, thermostats, pin locks. In following part, we show some of security flaws of most popular smart devices. We start with security assessment of smart thermostat, Google Nest, where researchers have revealed how it can be exploited and then used as smart spy device. Vulnerability required physical access to thermostat to load custom software over underlying Linux system. The compromised Nest thermostat acts as an entry point to attack other nodes within the local network exposing the data to the attacker [15].

Another device, where more realistic security issues were identified, is based on Insteon Hub, a network enabled device that can control light bulbs, wall switches, power outlets, thermostats, cameras and more. During a first Insteon Hub setup, user is asked to set up port forwarding from the Internet to the device. System documentation basically led user to expose access to anybody from the Internet. Furthermore, there was no option to enable authentication for the Web service running on the Insteon Hub that receives commands [16].

Smart home system with the most detailed security analysis is the Samsung propriety automation system called SmartThings [13]. Results revealed in SmartThings security analysis were trigger for this paper, and comparing the results similar set of issues is identified although the system is propriety [14]:

- SmartThings application can read all events a device generates if the application is granted at least one capability the device supports
- 42% of applications are granted capabilities that were not explicitly requested or used
- exploited framework design flaws to construct proof-of-concept attacks

On the protocol front, researchers demonstrated flaws in the ZigBee and ZWave protocol implementations for smart home devices [17]. Exploiting these bugs requires proximity to the target home but later can be used in attacks that do not require physical access to the home. Researchers also discussed the presence of some well-known vulnerabilities in home automation systems, such as Cross Site Scripting (XSS). Attacker could embed persistent Javascript in the log pages of one of the products. The researchers also observed that in some home automation systems, every communication between the homeowner and home automation system, both from within the home network and over the Internet, is done in clear text allowing an attacker to eavesdrop on the communication and gather legitimate login credentials [18].

III. OPENHAB SECURITY ANALYSIS

OpenHAB is designed in multilayer ecosystem with OSGi as baseline for its runtime engine showed on Fig 1. In this chapter we put focus on following components:

- openHAB Add-on Libraries
- openHAB REST service
- openHAB HTTP service

The Add-on libraries layer represents most vivid component of architecture also known as openHAB bindings. There are many bindings available, from house heating control to network binding allowing user to check, whether a device is currently available on the network. All bindings communicate with openHAB core through event based message bus. Next architectural component is HTTP service exposed as main management interface for openHAB configuration and administration. Every openHAB binding can be monitored, installed, or removed from this point. Last but not least important component is REST service. Most of the logic of REST service implements JavaScript Object Notation Application Programming Interface (JSON API) providing HTTP endpoints for third-party applications to interface with openHAB. This interface is main point for openHAB Android and iOS mobile applications to communicate

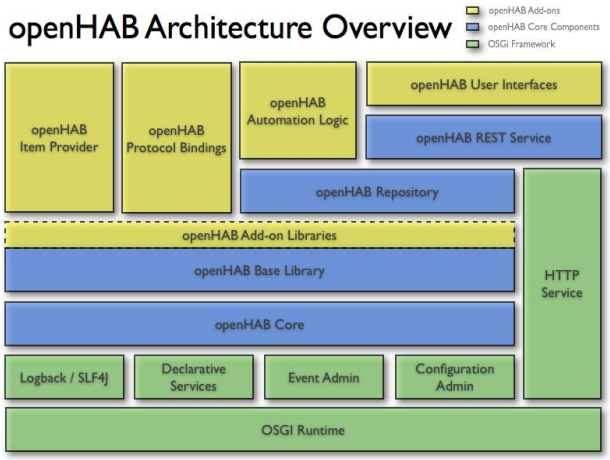

Figure 1. openHAB architecture [3]

towards openHAB event bus.

A. *Security analysis of core components*

To understand how the community based binding development manifest in practice from security point, we downloaded 153 [4] bindings from the openHAB official

web page and performed a static code analysis. We first present the number of bindings that are potentially vulnerable and then drill down to determine the extent to which apps are over privileged due to design flaws.

Table I shows the result overview of our dataset and most important that not all of these bindings are vulnerable. To show example, we pick few of these bindings to show actual vulnerability. Almost all bindings have some sort of Java coding bugs with resulting on its performance issues. Attaching higher number of bindings to openHAB runtime engine puts more focus on mentioned performance issues leading to process crash. After systems is crashed all openHAB functionality is terminated leading to smart home system which needs to be manually restarted. The same goes for openHAB REST service and mobile applications using it.

Adding more and more bindings to running system is very simple, but having in mind less code quality of some of bindings, could lead to openHAB process crash. From security perspective this is denial of service (DoS). However, this bug can be minimized by configuring openHAB in high availability mode (HA) where two instances are used. If one openHAB instance fails, other will take over and system will remain functional.

Next we show how we can utilize over privilege bindings to perform operations on system without permissions. One of widely used binding is openHAB Exec binding used to execute system scripts or commands directly from binding. Once started on system, this binding lets user to perform a command without any access control. The same goes for remote user using REST interface or using mobile application. Binding requires command in following structure:

```
exec="<[/bin/sh@@-c@@command]"
```

where command can be any shell command. To make a DoS showcase this can do remote system shutdown.

```
exec="OFF:ssh pi@192.168.1.4 shutdown -p
                    now"}
```

Exec binding is not installed by default, but is usually first choice for developers to do quick progress. This exact vulnerability is perfect example of user input validation problem we found by scanning bindings. Most of 153 bindings provide a user to control them by providing input arguments without any validation. Input from untrusted sources must be validated before use. Maliciously crafted inputs may cause problems, whether coming through binding arguments or external sources.

Another result analysis revealed is that some bindings uses direct communication with openHAB message bus. This is not an issue but shows that an attacker can create such binding which can listen for all events on openHAB message bus. Having complete access to all messages exchanged on bus can provide much information, such as security pins and passwords used in bindings connecting to external services such as Gmail Calendar. This kind of attack is not hard to achieve as all openHAB bindings are possible to download from web store where no cryptographic hash integrity functions are used by time of this analysis.

TABLE I. BINDINGS STATIC CODE ANALYSIS

Bindings count	153
Performance bugs	3019
Bindings with event bus access	9
User input not validated	103

OpenHAB exposes HTTP based web service using the Representational state transfer (REST) or RESTful protocol. REST-compliant Web services allow requesting systems to access and manipulate textual representations of Web resources using a uniform and predefined set of stateless operations [5].

This kind of service is widely used for many IoT systems as usage is very simple and does not require special programs to be used. RESTful service exposes several predefined operations that can be utilized to integrate openHAB with other systems as it allows read access to items and item states as well as status updates or the sending of commands for items. To access REST service of the running openHAB users simply browse link:

```
http://ip_address:8080/rest
```

where `ip_address` is IP address of machine running openHAB process. Complete REST schema is based on Swagger specification [6] supporting different media types. The REST service furthermore supports server-push, so one can subscribe client on change notification for certain resources. Reaching REST endpoint returns structured openHAB data such as:

- Bindings
- Binding Discovery
- Items
- Data persistence

Default configuration starts openHAB without any security activated at all. In general, it is advised to use HTTPS communication over HTTP. On the very first start, openHAB generates a personal (self-signed, 256-bit ECC) SSL certificate and stores it in the Jetty Web service keystore. This process makes sure that every installation has an individual certificate, so that nobody else can falsely mimic your server.

Maybe the worst thing regarding openHAB REST interface is that it does not support restricting access through HTTPS for certain users. There is no authentication in place, nor is there a limitation of functionality or information that different users can access. So, once someone has access to openHAB service it has access to all of its functionality. This looks like very bad approach, but the answer why relies on the fact that openHAB uses Eclipse Smart Home code as baseline [7]. Complete HTTP service part exposing REST interface is used from that code so the expected solution needs to be there.

Fortunately, there are few recommended workarounds to overcome this security vulnerability. First is to limit access and only allow REST requests coming from local loopback interface. The default value allows access from all interfaces:

```
web.listening.addresses = 0.0.0.0
```

and needs to be changed to:

```
web.listening.addresses = 127.0.0.1
```

to allow requests only from local machine.

Second solution is to run openHAB behind reverse proxy. A reverse proxy simply redirects client requests to the appropriate server. This means we can proxy openHAB request connections to other web service. Running openHAB behind a reverse proxy allows to access openHAB runtime via port 80 (HTTP) and 443 (HTTPS). It also provides a simple way of protecting server with authentication and secure certificates.

The last option is to setup account for openHAB Cloud service which enabled that openHAB runtime tunnels to it hiding all the communication. The main core features of openHAB Cloud are a user-management frontend, secure remote access, remote proxy-access, device registry & management, messaging services and data management & persistence. The openHAB Cloud also serves as core backend integration point for cloud-based features (e.g. IFTTT) and provides an OAuth2 application enablement [4]. Cloud openHAB will proxy home sitemap over HTTPS so the communication to UI is TLS encrypted.

To see how openHAB REST interface is vulnerable we used penetration testing tool OWASP Zed Attack Proxy (ZAP), an open-source web application security scanner intended to be used by both those new to application security as well as professional penetration testers [9]. When used as a proxy server it allows the user to manipulate all the traffic that passes through it, including traffic using https. It can also run in a 'daemon' mode which is then controlled via a REST Application programming interface.

Scan discovered four type of issues, two with medium severity (orange) and two with low (yellow) showed in Table II. First issue is related to application error disclosure. Some responses contain an error/warning message that may disclose sensitive information like the location of the file that produced the unhandled exception. This information can be used to launch further attacks against the web application. To solve this bug, review the source code of this page by implementing custom error pages. Consider implementing a mechanism to provide a unique error reference/identifier to the client (browser) while logging the details on the server side and not exposing them to the user. For example:

```
"error": {
  "message": "HTTP 404 Not Found",
  "http-code": 404,
  "exception": {
    "class":
"javax.ws.rs.NotFoundException",
    "message": "HTTP 404 Not Found",
    "localized-message": "HTTP 404 Not
Found"
  }
}
```

Second detected medium severity vulnerability is that X-Frame-Options header is not included in the HTTP response to protect against 'Click Jacking' attacks. Most modern Web browsers support the X-Frame-Options HTTP header. Ensure it's set on all web pages returned by your site (if you expect the page to be framed only by pages on your server (e.g. it's part of a frameset) then you'll want to use deny, otherwise if you never expect the page to be framed.

Going forward to low severity vulnerabilities we start with XSS protection not being enabled, or is disabled by the configuration of the 'X-XSS-Protection' HTTP response header on the web server. Posting data to REST API could make an attacker to post script which can lead to leakage of some user data. To minimize this kind of vulnerability openHAB web browser's XSS filter needs to be enabled, by setting the X-XSS-Protection HTTP response header to '1'.

The last but not the least important vulnerability is the X-Content-Type-Options header was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. As a solution administrator needs to ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages.

B. Exploiting system weaknesses

OpenHAB HTTP server is based on Java Jetty web service. Looking inside Jetty configuration it is noticeable that default configuration is used with no request filtering policy. Default HTTPS connection timeout is set to 30 seconds, giving easy way to produce high load request flooding and cause DoS. To do this test in this paper we used JMeter load generator software build by utilizing Java HttpClient object [8]. This object gives plain implementation of HTTP protocol without overhead resulting in high performance. Used generator is written in

TABLE II. OWASP ZAP SCAN RESULTS

Vulnerability Description	Count
Application error disclosure	8
X-Frame-Options HTTP header not set	829
Web browser XSSprotection not enabled	657
X-Content-Type-Options not set	829

asynchronous programming style, so that limited threads do not limit the maximum number of users that can be simulated. Generator is configured to start thread pool of 1000 parallel HTTP requests over the network as it gives more realistic performance and latency. Each thread represents unique user. The results revealed that 1000 parallel requests decreased openHAB availability by 30 %. Increasing number of parallel requests up to 10000 decreases availability to an extent that openHAB rest service cannot be used where 80 % of sent requests failed. To overcome this vulnerability default Jetty configuration needs to be updated by adding request filter. Filter places throttled requests in a priority queue, giving priority first to authenticated users and users with an HTTP session, then to connections identified by their IP addresses. The DoS filter limits exposure to request flooding, whether malicious, or as a result of a misconfigured client. The DoS filter keeps track of the number of requests from a connection per second. If the requests exceed the limit, Jetty rejects, delays, or throttles the request, and sends a warning message.

Proposed filter is shown here:

```
<filter
  <filter-name>DoSFilter</filter-name>
   <filter-class>DoSFilter /filter-class>
   <init-param>
     <param-name>maxRequestsPerSec</param-name>
     <param-value>30</param-value>
   </init-param>
 </filter>
```

where filter limits maximum number of requests per second to 30. The filter works on the assumption that the attacker might be written in simple blocking style, so by suspending requests you are hopefully consuming the attacker's resources. For a high reliability system, it should reject the excess requests immediately (fail fast) by using a queue with a bounded capability. The DoS filter is used to avoid Jetty thread starvation what is of great importance if openHAB runs on embedded system such as Raspberry Pi.

Earlier is showed that delivering openHAB bindings doesn't incorporate integrity validations, so all bindings downloaded and used directly could be potentially harmful. In this chapter we show how is easy to build an openHAB binding which could be used to expose system protected data over the REST interface. The purpose of a binding is to translate between events on the openHAB event bus and an external system. This translation should happen "stateless", i.e. the binding must not access the item registry in order to get hold of the current state of an item. Likewise, it should not itself keep states of items in memory. If proposed approach is not followed it is more likely binding will result in system resource starvation and potentially DoS.

To build openHAB binding skeleton we start with delivered project template to create OSGi structured Java project. Once created binding is automatically runnable as any other binding, but without any additional custom logic. For this purpose, to make openHAB event bus exposed, we need to add custom Java code to link event message bus and REST interface. First thing is to implement `BindingConfig` class over newly created Java class. Extending the class user can make own implementation of `processMessage` method with custom logic as all events happened on the message bus will pass this point. Logic depends on wanted result, and here we use any information gained on the bus by exposing it to remote REST service under our full control. Any content received on message bus will end up on our web service on the Internet and carefully stored. Working code is:

```
@Override
public void processMessage(String topic, byte[]
message) {

        URL url = new
        URL("http://our.page.com/logger.php");
        HttpURLConnection conn =
        url.openConnection();
        conn.setDoOutput(true);
        conn.setRequestMethod("POST");
        try {
           DataOutputStream wr = new
        DataOutputStream(conn.getOutputStream());
           wr.write(message);
        } catch(Exception e) {
        // hide potential trace
        }
}
```

where URL object defines remote logging service which will receive all the event data from custom binding. Remote logging service is plain PHP script which stores all data received by HTTP POST request in local file. Rest of the Java code does HTTP connection and writes every event bus message data to it. It is important to note that code catches all possible exception errors that could notify victim that something is wrong to hide suspicious operations.

Next step in this process is to build custom binding as Java archive library (jar) and put it in openHAB add-on directory. Attacker could potentially upload link to this binary to some of openHAB forums, but for test purposes we skipped this part and installed bindings directly on local instance. Once started binding automatically works and pushes all event bus messages to the remote logger script. To show extent of exposed data we were able to see if house was occupied, motion sensor state changes, security pin codes and much more. Part of logged file looks like following:

```
2016-08-06 18:12 [ItemStateChangedEvent] -
Motiondetector_Outdoor_Switch changed from
NULL to ON

2016-08-06 18:52 [ExecBinding] - executed
commandLine 'sudo ssh admin@192.168.0.106
shutdown -p now'

2016-08-06 19:16 [ItemCommandEvent] - Item
'Volume_Main_Bedroom' received command 0
```

C. *Binding usage best practices*

As is the case for openHAB binding repositories, further research is needed on validating available binding for smart homes. A language like Java provides some security benefits, but also has features that can be misused such as input strings being executed without prior validation. We need techniques that will validate openHAB bindings against code injection attacks, over privilege, and other hidden security vulnerabilities (e.g., disguised source code). For this purpose, automated code analysis can be performed by existing tools such as OWASP LAPSE+ or Java Security FindBugs. OWASP LAPSE+ is a security scanner for detecting vulnerabilities of untrusted data injection in Java EE Applications [10] and Java Security FindBugs is plugin for security audits of Java web applications [11].

To overcome intentional vulnerabilities openHAB community will need to introduce integrity validation of all community managed bindings. The other approach is to encourage, not to say force, users to migrate to openHAB Cloud service where installation of new bindings is performed by centrally managed bindings repository. Another solution is to let openHAB web service to validate used bindings and notify users if unofficial binding is found. Strict user confirmation needs to be enforced, similar as Android applications. Smart home devices and their associated binding software will continue to increase and will remain attractive to end user because they provide already developed functionality. However, the findings in this paper suggest that caution is required as well on the part of existing bindings, and on the part of binding designers. The risks are significant, and they are unlikely to be easily fixed via simple security patches alone.

IV. CONCLUSION

We performed a security evaluation of the popular open source home automation framework for programmable smart homes. No related work has been found available in this area, giving this paper first view over openHAB security issues. At the time of writing this paper results of analysis are not published, but can be retrieved upon user request to the authors.

Analysis of openHAB was a bit easier because all the framework and bindings code is accessible for any user. We performed a static code analysis to process existing bindings and determine how well the bindings quality is from programming perspective. We discovered (a) great number of performance issues in list of 153 existing openHAB bindings followed with incomplete documentation; (b) 9 bindings did use all the permissions by monitoring complete openHAB event bus potentially exposing over privilege to remote users; (c) more than hundred bindings accepts user input without prior validation; (d) 4 different security vulnerabilities, found by OWASP scan tool over openHAB REST service, are documented with proposed solution. We combined these open community flaws with other vulnerabilities and were able to do openHAB REST service load testing with DoS. Next area of interest was development of custom binding which can be utilized to proxy all event bus messages to remote service and monitor user lock pin-codes, motion sensor values, temperature, all without requiring permissions of user. The last part focused on areas of improvements to make openHAB community members more security aware to build ecosystem for future development.

REFERENCES

[1] M. Sripan, X. Lin, P. Petchlorlean and M. Ketcham, Research and "Thinking of Smart Home Technology," ICSEE, 2012.

[2] OpenHAB, http://docs.openhab.org/introduction.html, 20.01.2017.

[3] M.Porter, " Building IoT systems with openHAB," Konsulko.

[4] OpenHAB Bindings Wiki, https://github.com/openhab/openhab1-addons/wiki/Bindings , 10.06.2016.

[5] R. Thomas, "Architectural Styles and the Design of Network-based Software Architectures," University of California, 2000.

[6] T. Johnson, "Swagger tutorial for REST API documentation," http://idratherbewriting.com/2015/09/14/swagger-tutorial/ , 14.09.2015

[7] Eclipse Smart Home, http://www.eclipse.org/smarthome/documentation/index.html , 20.01.2017.

[8] N. Sravanthi, "Open Source Performance Testing Using Apache JMeter," CTS, 20.01.2017.

[9] Open Web Application Security Project, OWASP ZAP, https://www.owasp.org/index.php., 11.04.2017

[10] Open Web Application Security Project, OWASP LAPSE+, https://www.owasp.org/index.php/OWASP_LAPSE_Project 11.04.2017.

[11] Findbugs Security, http://find-sec-bugs.github.io/download.htm , 11.04.2017.

[12] J. Baker, "5 open source home automation tools", Red Hat, https://opensource.com/life/16/3/5-open-source-home-automation-tools , 28.02.2017

[13] Samsung SmartThings, http://www.samsung.com/uk/smartthings/ , 16.04.2017

[14] E. Fernandes, J. Jung and A. Prakash, "Security Analysis of Emerging Smart Home Applications", IEEE, 2016.

[15] G. Hernandez, O. Arias, D. Buentello and Y. Jin, "Smart Nest Thermostat: A Smart Spy in Your Home", Unversity of Central Florida, 2014.

[16] D. Bryan, "Home invasion 2.0", http://www.computerworld.com/article/2484542/ , 15.04.2017.

[17] A.J. Bernheim Brush, B. Lee, R. Mahajan, S. Agarwal, S. Saroiu, C. Dixon, "Home automation in the Wild: Challenges and Opportunities", SIGCHI, 2011.

[18] A. Cyril Jose, R. Malekian, "Smart Home Automation Security: A Literature Review", KAIS, 2015.