TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

David Lõssenko 179944IVSB

CYBER SECURITY AND DEVELOPMENT OF CROSS-PLATFORM APPLICATION THIRD-PARTY MODULARITY

Bachelor's thesis

Supervisor: Alejandro Guerra Manzanares
MSc

Tallinn 2020

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

David Lõssenko 179944IVSB

KOLMANDA OSAPOOLE PLATVORMIÜLESE RAKENDUSE MODULAARSUSE ARENDUS JA KÜBERTURVALISUS

bakalaureusetöö

Juhendaja: Alejandro Guerra Manzanares
MSc

Tallinn 2020

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: David Lõssenko

30.04.2020

Abstract

The aim of this thesis is to analyse cyber security concerns involved in the implementation of third-party modularity in cross-platform applications. This work gives a detailed analysis of the cyber security threats and different ways to mitigate them, while also providing a comprehensive overview of different technological stacks that could be used to implement a secure solution.

Third-party modules (also known as plugins or extensions) allow a third party to modify the business logic of an application to fit their own need. Implementing third-party modules is a security risk because it gives a certain portion of control over to a third party who might turn out to be malicious. This work presents caution and preventive measures that should be taken to mitigate the risks. More specifically, the preventive measures proposed in this work are: segregation of third-party modules and the application core on the level of software architecture, a centralized system of module distribution and a robust permission model. Finally, the paper covers the software solutions that could be used to address these issues while also providing an overview of how practical and secure those solutions are. The main result of this thesis is a set of guidelines for implementing secure third-party modularity in order to avoid their inherent cyber security related risks.

This thesis is written in English and is 39 pages long, including 5 chapters and 5 figures.

Annotatsioon

Käesoleva diplomitöö eesmärk on analüüsida küberturve muresi seotud kolmanda osapoole modulaarsuse implementeerimisega platvormiülestes rakendustes. Töö annab detailse ülevaate küberturve ohtudest ja erinevatest metoodikatest nende mitigeerimiseks. Samuti antakse põhjalik ülevaade erinevatest tehnoloogia pinudest, mida saaks turvalist lahendust implementeerides kasutada.

Kolmanda osapoole moodulid (teise sõnaga pluginad või pikendused) võimaldavad kolmandal osapoolel muuta rakenduse äriloogikat enda otstarveteks. Nende implementeerimine on turvarisk, kuna see annab teatud osa kontrolli eelnimetatud kolmandale osapoolele, kes võib osutuda pahatahtlikuks. Käesolev töö annab ülevaate ohutustest ja ennetatavatest meetoditest, mida tuleks jälgida, et riske vähendada. Täpsemalt, töös esiletoodud ennetusmeetodid on järgmised: kolmada osapoolte moodulite ja rakenduse eraldamine arhitektuuri tasemel, keskne modulaarne jaotus süsteem ja jõuline lubade mudel. Viimaseks, diplomitöö katab tarkvara lahendusi, millega saaks esiletoodud probleeme adresseerida, samaaegselt andes ülevaate nende praktilisusest ning turvalisusest. Eeldatav tulemus on kogum juhendeid turvaliste kolmanda osapoolte moodulite implementeerimiseks ning nende kaasaskäivad küberturberiskid.

Lõputöö on kirjutatud Inglise keeles ning sisaldab teksti 39 leheküljel, 5 peatükki, 5 joonist.

List of abbreviations and terms

TUT	Tallinn University of Technology
OS	Operating System
API	Application Program Interface
PWA	Progressive Web Application
TOS	Terms of Service

Table of contents

List of figures

Introduction

A third-party module (also known as plugin or extension) is a software component developed by a third party that adds a specific feature to an already-existing application. Examples of applications that utilize third-party modularity include, but are not limited to:

- Graphics software – could use plugins to add more functionality to image processing (Photoshop [1]).
- Media players – utilize plugins to add support for file types and custom filters (Winamp [2]).
- Integrated Development Environments – use plugins to add support for different programming languages or improve developer experience (Visual Studio Code [3], IntelliJ [4]).
- Game console emulators – use modules to enhance different aspects of hardware emulation (sound processing, graphics rendering, etc.) to enhance gaming experience (PCSX2 [5]).
- Web browsers – use a wide variety of extensions to enhance internet browsing experience (Mozilla Firefox [6]).

While implementing third party modularity for desktop-based applications is relatively simple and common, doing the same for mobile applications is much more difficult due to limitations implemented by the operating system (OS) creators that address cyber security concerns that might come up when using third-party software. The reason for the discrepancy of policies between desktop and mobile are due to the way that the distribution of applications is handled on the respective platforms. Desktop OS's, in their majority, don't utilize any kind of centralized system to distribute the software; instead, they mostly rely on installers that can be freely downloaded from the internet. Mobile devices, on the other hand, do have a centralized system of application distribution that

come in the form of application stores (Apple App Store for iOS and Google Play for Android).

From the cyber security's point of view, a centralized system of application distribution is more secure. The reason for that is because a centralized system provides the manufacturer of the OS a way to keep track of all the applications uploaded, which means that the manufacturer will be able to employ security audits for those applications and weed out the majority of the applications that are harmful to the user's device. The reason for why the desktop OS's are lagging behind in this regard is because they're older than the mobile OS's, and the idea of application stores is relatively novel. Applications stores became widespread with the rise of the smartphones, long after the personal computers (and by extension, desktop OS's) became popular.

Better security, however, comes with its trade-offs. For example, on iOS App Store, there is a very strong policy regarding application extensions. All of them need to be implemented via in-app purchases. This hinders the ability for a third party to implement custom functionality as all the in-app purchases are required to be first party. Those restrictions are in place because applications usually have access to the important parts of the OS (e.g. the file system), which makes third-party modularity a big security risk, as modules will be given access to those parts of the OS.

This work is about the analysis of various solutions for securing and implementing third-party modularity in cross-platform applications. Third-party modules, by their definition, are a very huge security risk and need to be approached with caution. In fact, vulnerabilities related to using software modules are in the top 10 security risks in 2019 according to the OWASP yearly report [7]. This work will overview different methods to address cyber security concerns related to third-party modularity, as well as provide an overview of possible technologies that could be used for implementation of secure third-party modularity to see which technological stack is optimal.

The topicality of this work and the need for a solution are related to opinions and complaints originating from members of communities gathered around some applications that have raised concerns over the absence of modularity on mobile devices. The outcome of this work should be applicable to any organisation, company or community that have

faced similar difficulties in finding a secure approach to implementation of third-party modularity.

The main questions that the author will be exploring during the thesis are as follows:

- What are the most relevant cyber security concerns that come up when attempting to implement third-party modularity?
- How can those cyber security concerns be addressed and solved? What are the compromises that could be made?
- What technologies could be used to implement cross-platform third-party modularity?
- What are the differences between those technologies and what caveats come with every choice? What is the most optimal solution for each of potential use cases?

With those questions getting answered, optimal approaches to solving the problem of security and implementations will be extracted. Approaches will differ depending on the degree of modularity that the developer wants the application to have. The approaches to the following use cases will be provided:

- The developer wants to implement secure third-party modularity for only a small part of the application.
- The developer wants the application to have as large degree of secure third-party modularity as possible.
- The developer wants no limitation on the degree of third-party modularity but will have a select few trusted third parties that could work on expanding the application's functionality.

The thesis will contain the following chapters:

1) Overview of the Cyber Security Concerns
2) Addressing the Cyber Security Concerns
3) Overview of the Software Development Solutions

4) Analysis of Software Solution’s Security and Practicality

5) Conclusion and Summary

1 Overview of the Cyber Security Concerns

1.1 Authentication

Most of the time, applications require the user to login in order to fetch user data from the back-end server. This could be done by user entering their username and password into a form, which then gets their credentials validated with the back-end server. After the credentials are validated, the user gets granted necessary access, which allows them to fetch necessary data. This is called authentication.

Authentication is usually done by employing security tokens, which get retrieved from the server after a successful login attempt. There are a few ways of doing that. Most notable solutions include JWT [8] and OAuth2 [9]. Those tokens are encrypted and usually contain information about the user's permission which decide the actions that the user is authorized to do. In order to minimize risks that come with token leakage, a token expiration and renewal mechanism is usually implemented. Secure tokens have an expiration time (usually, 1 hour) during which the client needs to request a token renewal. After retrieval of the token, the token gets included into every subsequent request to the back-end server (usually, the tokens are stored in the authorization header), through which, authorization of the request happens.

In a context of an application that allows third-party modularity, there is a risk that third-party modules might be sniffing the secure tokens out and thus making unwanted changes on behalf of the user. Moreover, by implementing third-party modularity, there is another risk that a misbehaving third-party module could read the contents of the login form and thus steal the password of the user.

A browser extension used by MEGA, a cloud storage and communication company [10], getting compromised back in September 4th of 2018 [11] is a popular example of an incident in which a compromised extension started stealing user passwords. The misbehaving extension stole user credentials from popular websites, including Github, Google, Amazon, Microsoft and more. The incident happened when an unknown attacker got access to the MEGA's Chrome Web Store account and uploaded a compromised

version of the extension. All the users of the extension who had the auto-update feature on Chrome enabled had their extension automatically updated to the compromised version. The extension was vulnerable for a total of 4 hours before MEGA team resolved the incident.

Securing the process of authentication is very important, as the process itself deals with very sensitive data, such as usernames, emails and passwords. Leakage of this data should be avoided as much as possible. In order to address security of the authentication process, the following concerns should be considered:

- Does the application store the tokens securely?
- Are third-party modules forbidden from accessing the token storage?
- Is the login form properly isolated from third-party modules?
- Is the process in which the token gets added to the authorization header properly isolated from third-party modules?

The implementation of secure authentication process in the context of an application that supports third-party modularity would require some crucial decisions on the level of software architecture. In this case, the third-party modules will have to be properly isolated from the part of the application that deals with authentication. This way, a misbehaving third-party module will not be able to steal the user data.

1.2 System Access

Applications usually interact with the device that they're being run on. That is done through utilization of the application program interface (API) provided by the operating system. The interactions with the device that an application might require include, but are not limited to:

- File system access – primarily used to preserve the state of the application between executions, by saving the crucial data into files.
- Utilization of native features – applications might require access to the native features (access to the video camera, motion sensors, battery usage statistics, etc)

- Internet access – besides being used for authentication, applications might have a need for data synchronization with the back end.

- Access to application data – sometimes, applications have use cases for collecting data from other applications (e.g., a fitness app might request data from the Google Fit to properly track workouts that the user has done throughout the day).

The implementation of third-party modules needs to be restrictive enough to limit the access to the system of the device, by allowing them access to only what's necessary. If proper precautions aren't met, then the application has a huge security risk of allowing unauthorized access to malicious modules. A malicious module that utilizes an insecure API might cause major damage to the user's device or stealing sensitive information. In order to prevent that from happening, the following concerns should be considered:

- What degree of system access does the application need to fulfil its function?

- Is the degree of which access to the system is given to the third-party modules sufficiently limited to only what is required?

- Are the system APIs properly isolated from the third-party modules?

- Are the users of the application given sufficient control over what access the third-party modules have?

With these concerns in mind, direct system access should not be given to third-party modules at all. Instead, an API that stands between the system API and third-party modules could be introduced that allows a limited access to the system calls. The degree of system access that any third-party module has should also be in full control by the user. The implementation of such an API should be planned on the level of software architecture.

1.3 Insecure Software Packages

Nowadays, software developers utilize help from third-party software packages in their projects in order to shorten the development time and difficulty. These packages are installed into the project via so-called package managers (e.g., NPM [12]). Package managers are usually command line interface tools that can install and setup software

packages by simply writing a command into the terminal. The target package then gets fetched from a global repository and installed into the project.

The use of package managers and third-party software packages has become a common place, so much so that it's very difficult to develop software without relying on one in some ecosystems. In very complex projects, the reliance on third-party software packages is very large. For example, Angular [13], a front-end framework developed and maintained by Google [14], has a component that depends on more than 950 software packages on NPM [15], some of which are first party, but most of which are third party. A very popular NPM package "core-js" [16], which is an extension to the standard library of JavaScript, gets over 20 million weekly downloads [17], and is used in a lot of projects.

Security issues that come with usage of third-party software packages are becoming more and more relevant nowadays with the widespread use of package managers. For example, on 11th of December 2019, a security vulnerability was publicly disclosed which affected all major package managers used by JavaScript [18]. The vulnerability allowed the attacker to publish a package onto the NPM repository that, when installed, could overwrite a global NPM executable that is used by another package, thus replacing the intended functionality of that executable with a malicious one.

Vulnerabilities related to package managers should not be taken lightly, as it could take only one popular third-party software package getting compromised to cause damage to thousands of other software projects that depend on it.

When talking about development of third-party modules for a cross-platform application, the probability that the third-party developers will utilize third-party packages in their own code base is significantly high, since currently that's the common practice. This makes it a sizable cyber security concern, as now a third-party module could itself turn out to be vulnerable if it happens to depend on a compromised third-party software package. From all of that, the following questions should be addressed:

- What are the possible package managers that could be used by the developers of third-party modules?
- What is the degree of support for those package managers? Does the degree of support warrant frequent security updates?

- How should awareness of potential risks be sufficiently communicated to the developers of third-party modules that might utilize software packages?

- In a case that a vulnerability does occur in a software package used by a third-party module, what is the best way to minimize risks and mitigate damage that could be done to the user of the application?

Similar to previous two sections, by implementing a restrictive software architecture around third-party modules, most of these concerns also get properly addressed. If a case of a bad software package usage does happen, since the third-party modules are limited with what they could do with the system, the risks will also get mitigated significantly. However, even if the damage that could be done to the device is limited, it is still possible to cause damage in some other way by disrupting the business logic of the third-party module itself. The risks could be further mitigated by introducing code audits for third-party modules as well as educating third-party developers themselves about the potential risks.

2 Addressing the Cyber Security Concerns

2.1 Secure Software Architecture

To address the cyber security concerns raised in previous chapters, the application will have to be designed in a way that isolates crucial operations, like authentication and system calls, from having direct access by third-party modules. This could be done by splitting the software architecture into two constituent parts: the application core and third-party module API.

Application core will handle all the tasks that would be too dangerous if given to third-party modules, like authentication and system access. The third-party module API is a layer that stands between third-party modules and the application core, which facilitates secure communication between the two parts. Third-party modules would then utilize the API layer to make changes to the system of the device in a controlled and secure manner.

The software architecture pattern that fulfils the aforementioned requirements the most is the microkernel pattern (also known as the plug-in pattern). The architecture of the application is illustrated on the Figure 1:

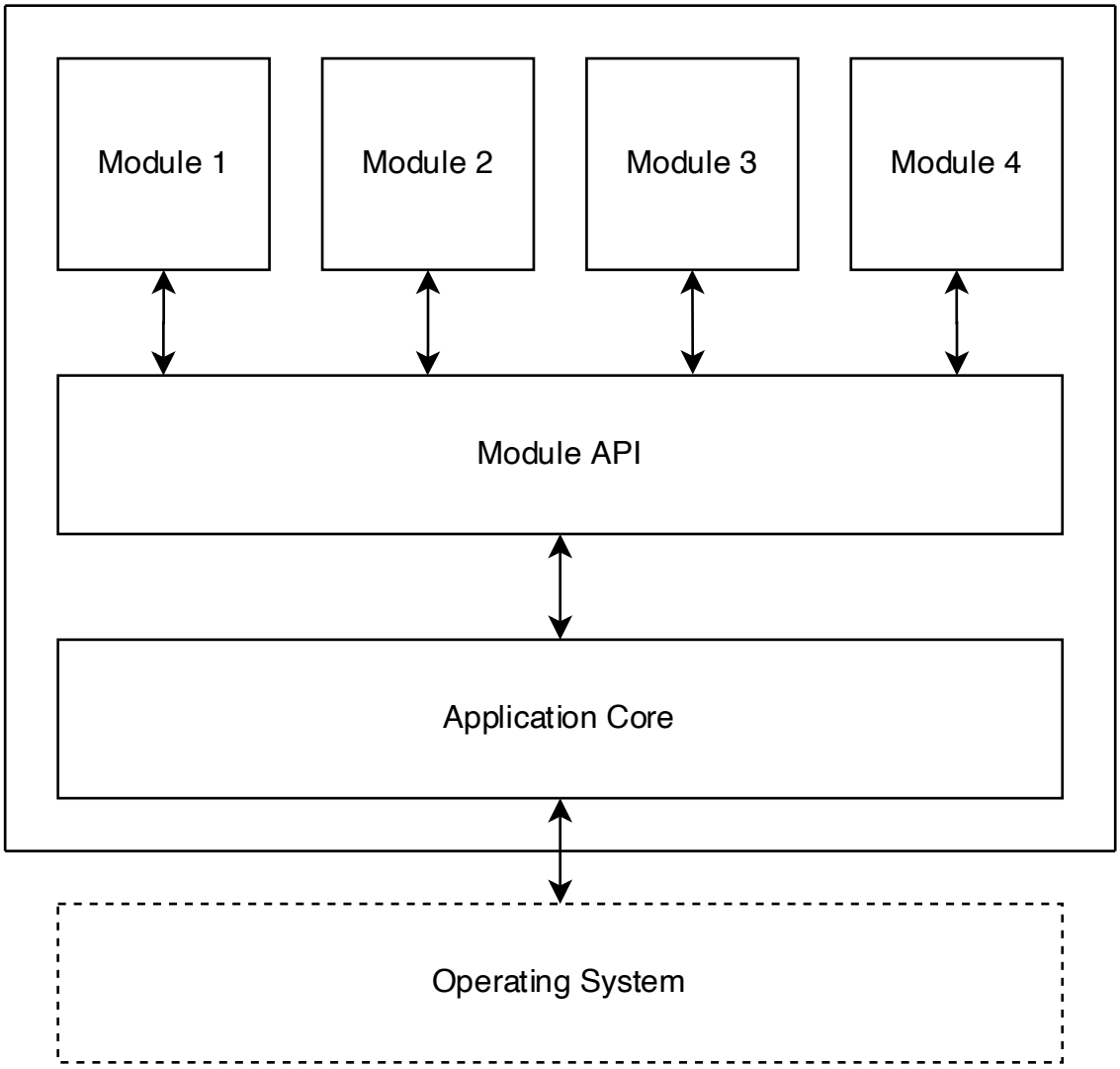

Figure 1. Software Architecture of the Application (Source: Created by the author)

By utilizing this architecture, the application core is properly isolated from third-party modules. Since the application core is the only component of the architecture that could affect the operating system, the access to it should be properly limited. The only component that has any degree of control over the core is the module API. The API layer ensures that the interaction between the core and modules could be limited to only allow the degree of control intended by the developer of the application.

With this separation defined, the following parts of the business logic should be delegated to the application core:

- Authentication – this includes the authentication form, token expiration/renewal logic and in-memory storage of tokens. By making authentication a part of the core,

it will then be fully isolated from the modules. The API should not provide any access to it whatsoever.

- System access – this includes access to the file system, native features, and network. By making all of this a part of the core, the API will be able to limit the scope of system access of third-party modules as seen fit by the developer.

- Third-party module management – the user has to have full control over what modules should be installed and activated; moreover, the user should have full control over what kind of permissions any specific third-party module has. Based on the granted permissions, the module API will then limit access to the application core as needed. To prevent third-party modules from being able to have any effect on what modules are activated or the corresponding permission of said modules, they should not be able to affect the module management part of the application.

With all of that in mind, most of the concerns brought up in the previous chapter over authentication and system access are addressed. Moreover, the provided software architecture also minimizes the risks that come with third-party developers utilizing software packages that are potentially insecure. Since the scope of what the third-party modules could do is limited to what the API and the corresponding permissions allow, the cyber security risks are significantly mitigated, albeit not fully.

2.2 Module Management

Implementing third-party modularity means that the user of the application will have the ability to utilize third-party modules to customize the logic of the application in a way that would fit their needed use case. For that, the developer has to provide the user with a way to install and uninstall third-party modules. This could be done with either the in-app interface or by external means outside of the application itself. Usually, this interface is called a plugin store or a module store.

For example, an external module management system could be utilization of the official website to host the module store where the user could choose which modules they want to use. In-app module management could include a user interface in which the user could turn the third-party modules on and off; and as stated above, that part would have to be the part of the core.

After authentication inside of the application itself is done, the application will then request the active modules from the back end and load them. The active modules will then modify the business logic of the application to the extent that is allowed by the permissions given to any specific third-party module.

Developers of third-party modules could publish their module onto the module store, which would require approval of the developer of the application itself. Depending on the budget of the application developer, a code audit process could be employed to further minimize the risks that come with the implementation of third-party modules. In such a scenario, every consequent release would then have to go through code audit over again. Once the third-party module is approved, it will then be publicly published to the module store where users will be able to install them into their own application.

In order to minimize the cyber security threat of third-party module developer's accounts getting stolen, a two-step verification could be enforced on every account that wishes to publish a third-party module. Moreover, educating third-party module developers about the risks that could come with using software packages in their projects could raise awareness within the community of developers and thus further mitigate risks that come with software package usage.

2.3 Access Permissions

The module store should give the user a clear idea about what exactly any specific third-party module is doing. This could be done by implementing a certain set of permissions that third-party modules could be granted. Upon the installation of the module, the user will have to consent to granting those permission to the third-party module. This way, the user will be able to detect whether the module is doing more than it initially advertised.

The permission model for the modular part of the application will differ depending on what exactly the function of the application is. Some applications, like simple calendar apps, might choose to have a smaller set of permissions as they have no use for native features like camera or sensors. Other applications, like fitness-tracking applications, could utilize the motion sensors to track the workout done for the day. In that case, they would have a larger permission model because the application itself utilizes more data.

The permissions in the permission model could include:

- Filesystem access – if the module needs access to the filesystem to, for example, retain useful data or access the data of the core application. This permission could be further divided into two categories:
    - Sandboxed access – the third-party module is given access only to a single folder created just for that module, with no access to the filesystem outside of that folder.
    - Application-wide access – the third-party module is given access to all the data and files in use by the application itself.
    - Broad access – the third-party module is given broader access to the device's filesystem. This might include the user's photos, videos, downloads, etc. This option could be divided further to limit the access of the module so that it could only have what's absolutely necessary.
- Internet access – if the third-party module needs to connect to the internet to, for example, communicate with a third-party API to fetch or upload necessary data. This could be limited to only allow HTTP/HTTPS requests but could be expanded further.
- Access to the native features – if the third-party module requires access to a native feature like camera or motion sensors. This option could be broken down to have a separate permission for each of the native features.
- Access to other applications – sometimes access to another application will be needed. That would be required if data from another application is required to utilize the task that is done by the specific third-party module. In this case, access to each application will be treated as a separate permission.

The list of permissions could be expanded further depending on the use case. The more permissions there is, the more control the user will have over what a third-party module is doing. By listing those permissions, the user will be aware of what to expect, and if they suspect something, they can simply choose to not grant this permission to the third-party module.

3 Overview of the Software Development Solutions

Today, there exists a multitude of different frameworks for development which could target multiple platforms. The most known examples are NativeScript [19], React Native [20], Flutter [21] and Xamarin [22]. These frameworks have their pros and cons when it comes to achieving third-party modularity while employing the secure software architecture devised above.

3.1 NativeScript and React Native

NativeScript is a framework for building native applications with Angular, Vue.js [23], TypeScript or JavaScript. It takes the TypeScript or JavaScript codebase and makes it possible to be run on iOS or Android with the full access to their respective API's. This allows for the business logic to be the same for all release targets, with differences existing only when the application interacts with the API's of either platform. The framework's logic and mechanism is illustrated on Figure 2.

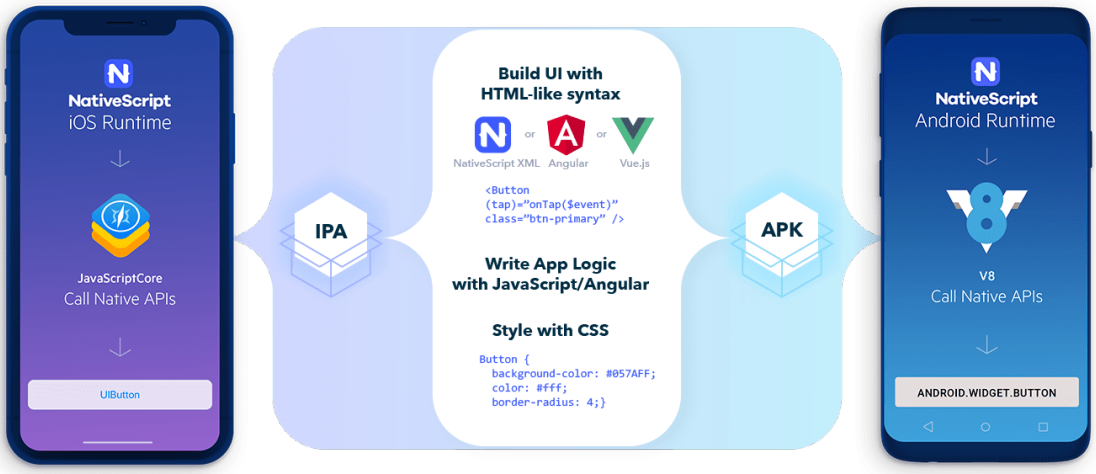

Figure 2. Mechanism of NativeScript (Source: https://d2odgkulk9w7if.cloudfront.net/images/default-source/home/how-it-works-min.png)

React Native is a framework with the same idea as NativeScript, but it's based solely on the development flow of React.js [24] (hence the name). React components are wrapped around the existing native code that then interact with native API's, as illustrated on

Figure 3. The framework being based on React.js means that other front-end web frameworks or technologies cannot be used with it.

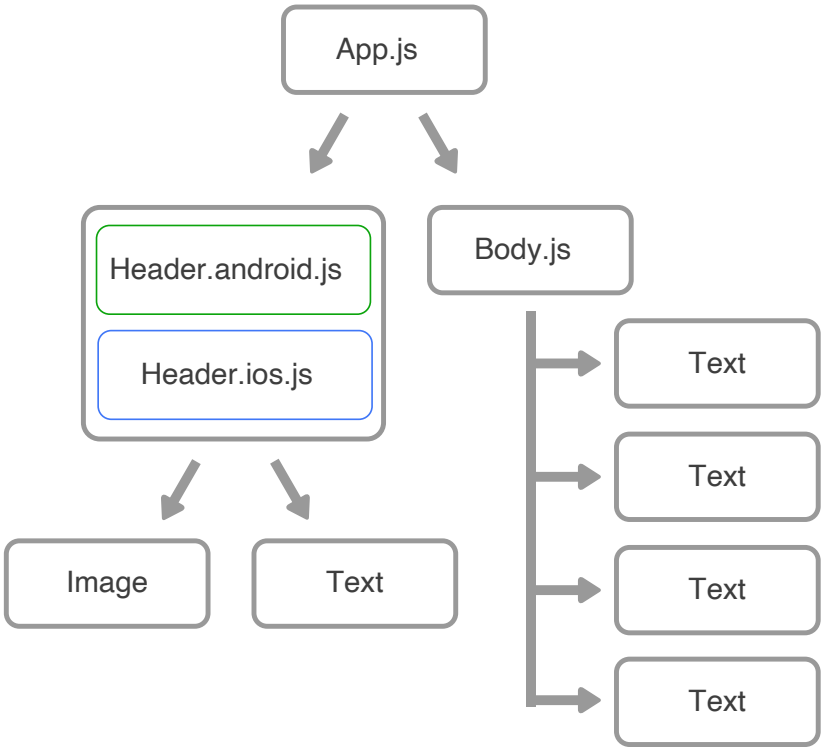

Figure 3. Diagram Illustrating how Native Code is Wrapped into React Native Components (Source: https://reactnative.dev/img/homepage/cross-platform.svg)

The frameworks, however, do not yet target desktop, which makes it somewhat difficult for applications to be built for desktop. It's still possible, but it requires a tricky integration with Electron [25] so that the business logic of the application could be run on desktop. Recently, the development team behind NativeScript has announced that they're experimenting with adding support for targeting desktop, but as of writing of this thesis, there hasn't been any working prototypes presented to the public just yet.

Since the frameworks have JavaScript at the core of the development, it opens up a myriad of possibilities for implementing third-party modularity.

3.1.1 Third-Party Module API in JavaScript

One of the possibilities, is to create a third-party module API in JavaScript itself. Since the application is already written in JavaScript, there is no necessity for integrating an interpreter. All things considered, it's a very big security risk, because the third-party JavaScript must be quarantined as to not be allowed to access the application core, as the application core itself would also be written in JavaScript and will need to be properly

isolated from third-party modules. This would be a difficult feat to achieve as the third-party module code will be getting executed on the same execution thread as the application core.

Google has made a library specifically for this purpose. It's called Caja [26]. It lets web applications to run third-party code in a quarantine, while exposing only a very limited (but configurable) API to the third-party code itself. However, it only supports versions of JavaScript up to ES5 [27], a version of JavaScript that is considered to be outdated by today's standards. Another way of achieving a quarantine would be to use JavaScript interpreters running in JavaScript, something like Sval [28], which was made exactly for this purpose. It does, however, mean, that the third-party JavaScript will be run considerably slower, as there is a big overhead for a quarantine like that.

Looking from the viewpoint of cyber security, a conclusion could be derived that making third-party module API in JavaScript in the context of the application core being developed using either NativeScript or React Native is not desirable, as that could lead to security issues. The performance will also suffer with this setup.

3.1.2 Third-Party Module API in Other Scripting Languages

Since using JavaScript for implementing third-party module API is a security risk if frameworks like Native Script or React Native are used, using other scripting languages could be preferable. Thankfully, JavaScript is a vastly popular programming language, which makes finding interpreters for other languages written in JavaScript a lot easier.

A scripting language by the name of Lua [29], which is a very efficient, light and embeddable scripting language, could be used as the scripting language of choice for the third-party module API. Lua is a simple language that doesn't take too much time to learn and can be quickly setup. A project called Fengari [30] implements a full feature set of Lua in JavaScript, and is currently being maintained, which makes this a good choice.

For something more popular, a scripting language like Python could be used. A project by the name of Brython [31] allows Python scripts to be run from within JavaScript. It is currently being well maintained and could serve as a good alternative.

As far as cyber security is concerned, using a scripting language besides JavaScript for implementing third-party module API is preferable. By using a scripting language

interpreter, it makes it a lot easier to control and properly isolate third-party modules. That is because the third-party module API itself could be directly controlled by the application core, which in turn makes it more secure.

3.2 Flutter and Xamarin

Flutter is a mature framework for building cross-platform applications. It uses Dart as its primary programming language and has a large toolset aiding the development process. Unlike NativeScript and React Native, Flutter does support desktop targeting, albeit desktop targets lack some feature set. Flutter is in active maintenance by Google, which makes it a safe bet. On top of all that, it's fully free and open source.

Xamarin is another mature framework for building cross-platform application. It uses C# and .NET stack [32] at its core. Its toolset is also considerably large, and just like Flutter, it can also target desktop right out of the box, and its desktop feature set is more mature than Flutter's. The framework is being actively developed and maintained by Microsoft. It is, however, not free to use, and will require licensing for bigger enterprises. It's architecture is illustrated on Figure 4.

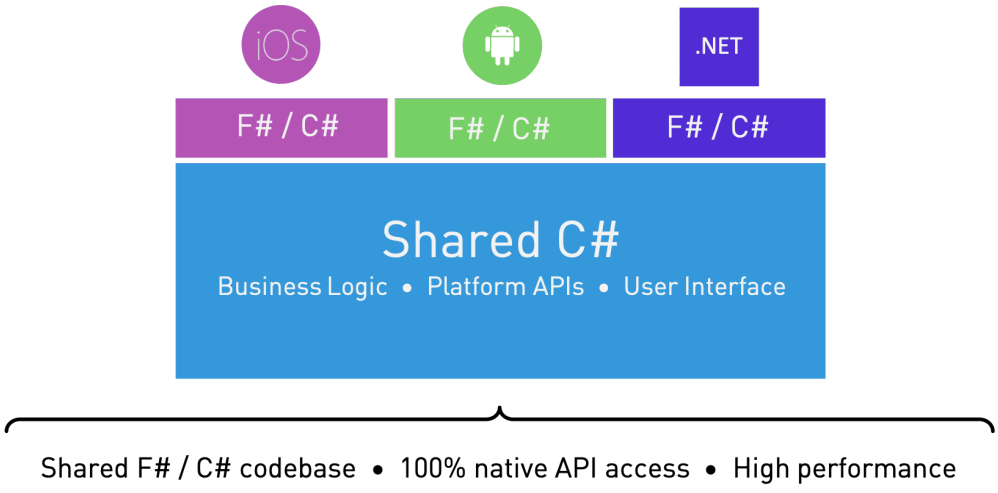

Figure 4. Xamarin Architecture (Source: https://hjerpbakk.com/img/bekk-christmas/xamarin-forms-light.png)

Both Flutter and Xamarin are compiled into native applications, which would make modification to the core functionality by the third party rather difficult. Thankfully, considering that the frameworks are compiled, it will make using interpreted programming languages a lot faster, as then the developer of the application wouldn't

need to worry about the unnecessary overhead that would exist for NativeScript and React Native.

It can be, however, difficult to implement an interpreter for a scripting language. In this case, the developer of the application will have to find and compile the appropriate libraries for each platform, as well as keep them up to date. This proves to be difficult, as there can be platform-specific problems which could then be hard to resolve. In spite all that, both platforms provide web view elements, which do run JavaScript out of the box. Nevertheless, it can be difficult to provide a robust API for third-party modularity using web views, as then the business logic of the application will have to be partially implemented using web technologies, and in this case, it would lack access to native features and system access.

3.3 Progressive Web Applications

A progressive web application [33] (PWA), is an application that is built using web technologies that can be used in the same manner as a native app. Basically, it's just a normal webpage that acts as if it were a fully-fledged, offline application. PWA's don't need to be installed (as the contents of a PWA are required from a webserver). When used offline, a service worker is used to fetch cached webpage so that the application can be used offline. PWA's can be used to target both mobile and desktop. Using a PWA gives an additional advantage of being as secure as using normal web browsers. Which, of course, can also be limiting in a lot of ways.

PWA's utilize what's known as a service worker as a middleman between the front end and the back end. The service worker caches webpages and API calls, allowing the webpage to function offline after the initial visit. Service worker also interacts with the native platform in a way that allows the application to use native features. The mechanism and logic of the service worker is illustrated on Figure 5.

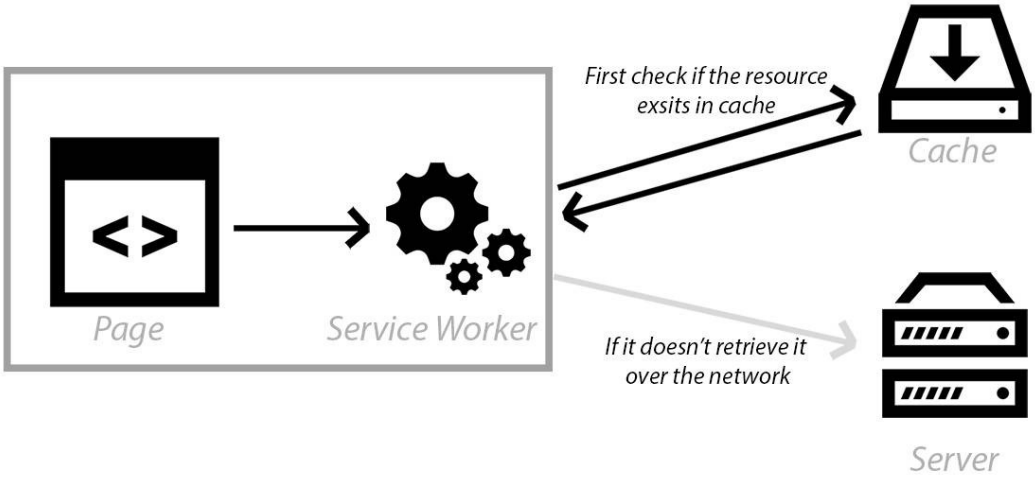

Figure 5. Service Worker Mechanism (Source: https://miro.medium.com/max/2000/0*dVlfyYWOH1GnGNeL.jpg)

The technology has been primarily developed by Google and sees the biggest support on Android devices. Despite that, Apple has been working on adding support for PWA's to their platforms as well. While the support for PWA's on iOS is currently limited, it is still very much functional. Using PWA's, however, can bring about difficulties when it comes to using native features. Accessing native features on Android is a lot easier to achieve. iOS, on the other hand, still lacks some features in that regard.

Implementing a third-party modularity with PWA's would be akin to achieving it with NativeScript or React Native, where a quarantine or an interpreter could be used for execution. Since PWA's also follow security standards of normal web browsers, it could also be a safer choice, as system access is limited.

3.4 Limitation of iOS App Store

In the terms of service (TOS) of the App Store [34] it is stated, that any and all modifications to the current functionality of an application shall only be done through in-app purchases. That clause single-handedly forbids third-party modularity from being implemented into an application that runs on iOS. This is quite a big roadblock towards achieving full cross-platform modularity. This is one of the reasons for why the Firefox browser for mobile only support extensions for Android, but not for iOS [35]. There are, however, a few ideas that might help in overcoming this roadblock.

3.4.1 Module Bundling

One way of achieving this could be bundling all the modules alongside the release of a new version of an application. This would solve the issue of expandability during the runtime of the application. This approach has quite a few caveats, however. Firstly, bundling all plugins together is quite inefficient as it will increase the overall size of the application. Secondly, each update of a plugin by a third party will have to require a new version release. Finally, the development process of an application will have to take the development processes of all the third party into consideration, so that the updates to both the main application and the third-party modules could be synchronized properly. These problems end up creating a lot of difficulty to warrant introducing modular design into the application.

This solution could be acceptable if the developer of the application wishes to have a select few third-party developers that are allowed to develop third-party modules. This approach reduces risks of a third-party module going rogue as the developer will have full control over who is allowed to develop third-party modules. Moreover, by having a trusted list of select few third-party developers, enforcing audits and development standards will also become easier as the number of third parties is kept small. This solution, however, is not suitable for applications that want to achieve a full-scale third-party modularity with as much freedom as possible.

3.4.2 Web Views

By isolating certain areas or components of the application into web views, their functionality can then be modified by a JavaScript-based third-party module API. This would solve the problems with TOS. Web views allow third party code to be executed, because they follow the security standards of web browsers, and therefore meeting the requirements presented by the TOS. However, as mentioned in earlier sections, this would limit the capabilities of third-party modularity. The developer of the application will have to delegate some logic to the web views (and by definition, implement them using web technologies) to achieve the needed degree of modularity. This solution is acceptable if the degree of modularity that the application needs is not large scale. This solution, however, will not be acceptable for applications that want as high of a degree of modularity as possible.

3.4.3 PWA's

Since PWA's are basically websites running on a standard web kit of the respective platform, the rules surrounding PWA's are the same as any other web application, meaning that it follows the security rules of web applications, and not native applications. PWA's don't require to be installed from App Store, and therefore don't require to pass the requirements of a native application. PWA's can be put on the user's home screen without needing for an installation procedure.

As it currently stands, if the developer wants to include a greater degree of modularity to an application and make it work on both iOS and Android, PWA is the only conceivable way to go about it. Using a PWA also gives an added benefit that the majority of the code base could also be reused if the developer wants to have a browser version of the application (albeit, it will lack native features).

4 Analysis of Software Solution's Security and Practicality

4.1 Degree of Third-Party Modularity

Degree of modularity is the degree of freedom that any third-party module could have to modify the business logic of the application. Depending on the application, degree of third-party modularity could vary. Some developers might want to make it so that their applications would be as modular as possible, making sure that the business logic of the application itself could be changed by a third-party module to the greatest degree possible. Other developers, on the other hand, might want to make only one specific part of an application modular. It is very important that the developer makes an educated decision on how much freedom should be delegated to third-party modules and plan software development around that.

From the cyber security's point of view, the less modular the application is, the more secure it will be from the abuse by misbehaving third-party modules. By limiting the scope of what a third-party module could do, it also limits the scope of abuse. The developer should implement the degree of modularity to only what is absolutely necessary and is required. Depending on the desired degree of third-party modularity, an appropriate decision on the technological stack could then be made.

4.2 Highest Degree of Third-Party Modularity

In case of the developer wanting to implement as high of a degree of third-party modularity as possible, then the choice will depend on whether the developer wants to support iOS. Depending on that decision, an appropriate software solution could be derived.

If the developer wants to support iOS, then the only choice for achieving the highest degree of modularity possible is by utilizing PWA's. Since third-party modularity is forbidden by the Apple App Store's TOS, a native app can't support a high degree of modularity on iOS. However, considering the fact that PWA's act a little bit differently

compared to the traditional native applications and follow the security measures of browsers, by utilizing a PWA, a high degree of third-party modularity could be implemented. On top of that, since the PWA's follow the security standards of normal browsers, they are naturally more limited in scope to what they could do to harm the device of the user.

In case of developer wanting to drop the iOS support, then that opens up the opportunity for more choices. In this case, a native framework that uses JavaScript (NativeScript or React Native) could be used alongside an interpreter for implementing the third-party module API. These frameworks are easy to pick-up and develop with, but unfortunately introduce unnecessary overhead because they use an interpreted language (JavaScript). This could be unnoticeable for applications that don't do anything computationally intensive. A choice of framework that would be faster are Xamarin and Flutter. These frameworks compile straight to native code and therefore will run a lot faster in comparison to native JavaScript frameworks. However, these solutions will not be able to support third-party modularity on iOS devices.

4.3 Limited Degree of Third-Party Modularity

The developer might want to introduce a modular part to the application, but not necessarily make the whole application modular. This part could be either a component or a set of components the business logic of which could be modified by a third-party module. In this case, native JavaScript frameworks and frameworks like Xamarin and Flutter could be made to work in compliance with Apple App Store. All of the aforementioned frameworks support webviews, which makes it possible to delegate the logic of some components to webviews. Essentially, making "mini-websites" and displaying them as components. This is possible but could be proven difficult as then the business logic of the application will be segregated between two technological stacks (native application code and web code). But nevertheless, this would allow for publishing a native application that supports a limited degree of third-party modularity to the Apple App Store.

Another way of achieving this could be module bundling. Basically, shipping all of the modules already bundled with the main application. In this case, the developer could have a select few trusted third parties that are responsible for developing those modules. This

approach is most secure and works on all platforms. In this case, any of the previously mentioned solutions could be used. Since the number of the trusted third parties is limited, a code audit could be introduced that ensures the security of those modules. The caveat is that the application size will be large as now all the third-party modules are bundled, and the user of the application might not use all of them.

4.4 Conclusions about Cyber Security

So far, it's been established that the preferred technological stack for implementation of cross-platform third-party modularity will depend on the degree of modularity that the developer needs to implement. As stated previously, the smaller the degree of modularity is, the more secure the application will turn out in the end. This section will analyse all the technological stacks from the viewpoint of cyber security.

4.4.1 Module Bundling

Module bundling for limited degree of third-party modularity combined with a select few trusted third-party developers is by far the most secure approach when a large degree of third-party modularity is not needed. In this case, the employment of mandatory code audit would require less resources to achieve, as the number of trusted third parties is small.

Moreover, with this approach the degree of modularity could be expanded upon the request of third parties. Since the third parties are trusted, a higher degree of system access could be granted to them if the demand requires so. In order to further minimize the risks, a 2-step verification could be added into the process of authorization for the trusted third parties. This minimizes the risks of third-party modules getting hi-jacked.

4.4.2 PWA's

A PWA is the most secure way to go in a scenario which demands as high of a degree of modularity as possible. Not only will it work on all platforms, it also provides robust security measures implemented by the browsers. Which means that the system access is already very restricted to begin with.

Since PWA's are not limited to any single framework or front-end technology, this opens for a possibility to utilize any JavaScript framework there is. In some cases, a framework

might be preferable over another due to security issues in the latter. On top of that, a larger domain of tools will be available to the developer to resolve security issues that might arise during the development process.

4.4.3 Native Frameworks

NativeScript and React Native could also be good candidates for developing a secure cross-platform application that supports third-party modularity. In this case, an interpreter for a scripting language could be utilized that will implement the third-party module API. JavaScript in a sandbox could be used as well, but it comes with a variety of cyber security issues, because the third-party module code and the main application (core) code will be run on the same process, which could make the isolation of third-party modules difficult. From the viewpoint of cyber security, an interpreter is preferable over just using JavaScript in a sandbox.

Xamarin and Flutter are also very good candidates and will also require a scripting language interpreter to come with it. As far cyber security is concerned, it’s more or less just as secure as either NativeScript or React Native. The real difference is that since Xamarin and Flutter utilize a compiled language, they will overall be more performant over the JavaScript frameworks.

5 Summary

The goal of this thesis was to analyse cyber security concerns that come with implementing third-party modularity in cross-platform applications, to find ways to mitigate the security risks and then to compare possible software development solutions. The author explored cyber security concerns related to authentication, system access and software package usage.

In the analysis, the author explored ways to mitigate cyber security concerns. A proposal for a secure software architecture and best practices was made, which was then followed by an analysis of possible software development solutions. Best practices that would mitigate the cyber security risks are as follows:

1) Proper segregation of third-party modules has to be implemented on the level of software architecture. By doing so, the interaction between the OS and third-party modules will be limited to the extent that is allowed by the third-party module API.

2) Centralization of third-party module distribution allows for easier utilization of code audits which will help filtering malicious modules out. Moreover, a two-factor authentication could be imposed on third-party developers which would reduce risks of their accounts getting stolen and third-party modules turning malicious.

3) A secure permission model that lets the user decide which permissions each third-party module has will allow users to have more control over them. On top of that, having a permission model will also raise awareness of the user about what any third-party module might be doing.

Based on the analysis a research was conducted comparing various pros and cons of the software solution proposals from the viewpoint of cyber security and practicality alike. Based on the research, the author devised the optimal software solutions for the following use-cases:

1) Best for implementing a limited degree of third-party modularity – Xamarin, Flutter, React Native or NativeScript

2) Best for implementing as high of a degree of third-party modularity as possible – PWA

3) Best for implementing third-party modularity with a select few trusted third parties – module bundling combined with any of the native frameworks

Analysis of the software solutions had the following conclusions about cyber security:

1) Module bundling with a select few trusted third parties is the most secure solution, as employing code audits will require considerably less resources when the total amount of third-party modules is small.

2) PWA is a secure choice for developing third-party modularity because they follow the security standards of the web browsers and have a restricted access to native features and system access that is harder to abuse compared to native applications.

3) Native cross-platform frameworks have a good security model as well but have a lot less restricting access to both the system and native features, which is easier to abuse by malicious third-party modules.

In general, the lesser the degree of third-party modularity is, the more secure the application will end up being. The author recommends implementing a degree of modularity up until the point that is absolutely necessary. Not all applications require the highest degree of modularity, which means that establishing limits from the very beginning of the development process will result in a more secure application overall.

References

[1] Adobe Inc., “Photoshop,” 2020. [Online]. Available: https://www.adobe.com/products/photoshop.html. [Accessed 29 April 2020].

[2] Radionomy, “Winamp,” 2018. [Online]. Available: https://winamp.com/. [Accessed 29 April 2020].

[3] Microsoft Corporation, “Visual Studio Code,” 2020. [Online]. Available: https://code.visualstudio.com/. [Accessed 29 April 2020].

[4] Jetbrains s.r.o, “Intellij IDEA,” 2020. [Online]. Available: https://www.jetbrains.com/idea/. [Accessed 29 April 2020].

[5] PCSX2 Team, “PCSX2,” 2020. [Online]. Available: https://pcsx2.net/. [Accessed 29 April 2020].

[6] Mozilla Corporation, “Mozilla Firefox,” 2020. [Online]. Available: https://www.mozilla.org/en-US/firefox/new/. [Accessed 29 April 2020].

[7] OWASP Foundation, “Top 10 Web Application Security Risks,” 2020. [Online]. Available: https://owasp.org/www-project-top-ten/. [Accessed 29 April 2020].

[8] Auth0 Inc., “JWT,” 2020. [Online]. Available: https://jwt.io/. [Accessed 29 April 2020].

[9] Aaron Parecki, “OAuth 2.0,” 2020. [Online]. Available: https://oauth.net/2/. [Accessed 29 April 2020].

[10] Mega Ltd., “MEGA – The Privacy Company,” 2020. [Online]. Available: https://mega.nz/about/main. [Accessed 17 April 2020].

[11] US Medical IT, “Another Chrome Extension Is Stealing Passwords,” 2018. [Online]. Available: https://www.usmedicalit.com/2018/09/18/another-chrome-extension-is-stealing-passwords/. [Accessed 17 April 2020].

[12] NPM Inc., “NPM,” 2020. [Online]. Available: https://www.npmjs.com/. [Accessed 29 April 2020].

[13] Google LLC, “Angular,” 2020. [Online]. Available: https://angular.io/. [Accessed 29 April 2020].

[14] Google LLC, “About Google,” 2020. [Online]. Available: https://about.google/. [Accessed 29 April 2020].

[15] Mmorszczyzna at Hackermoon, “What’s really wrong with node_modules and why this is your fault,” 2017. [Online]. Available: https://hackernoon.com/whats-really-wrong-with-node-modules-and-why-this-is-your-fault-8ac9fa893823. [Accessed 8 April 2020].

[16] Denis Pushkarev, “core-js,” 2020. [Online]. Available: https://www.npmjs.com/package/core-js. [Accessed 8 April 2020].

[17] John Potter, “NPM Trends – core-js,” 2020. [Online]. Available: https://www.npmtrends.com/core-js. [Accessed 8 April 2020].

[18] Liran Tal, “Understanding filesystem takeover vulnerabilities in npm JavaScript package manager,” 2020. [Online]. Available: https://snyk.io/blog/understanding-filesystem-takeover-vulnerabilities-in-npm-javascript-package-manager/. [Accessed 8 April 2020].

[19] Progress Software Corporation, “NativeScript,” 2020. [Online]. Available: https://www.nativescript.org/. [Accessed 29 April 2020].

[20] Facebook Inc., “React Native,” 2020. [Online]. Available: https://reactnative.dev/. [Accessed 29 April 2020].

[21] Google LLC, “Flutter,” 2020. [Online]. Available: https://flutter.dev/. [Accessed 29 April 2020].

[22] Microsoft Corporation, “Xamarin,” 2020. [Online]. Available: https://dotnet.microsoft.com/apps/xamarin. [Accessed 29 April 2020].

[23] Evan You, “Vue.js,” 2020. [Online]. Available: https://vuejs.org/. [Accessed 29 April 2020].

[24] Facebook Inc., “React.js,” 2020. [Online]. Available: https://reactjs.org/. [Accessed 29 April 2020].

[25] GitHub Inc., “Electron.js,” 2020. [Online]. Available: https://www.electronjs.org/. [Accessed 29 April 2020].

[26] Google LLC, “Caja,” 2019. [Online]. Available: https://github.com/google/caja. [Accessed 29 April 2020].

[27] Google LLC, “Benefits of using Caja,” 2020. [Online]. Available: https://github.com/google/caja#benefits-of-using-caja. [Accessed 29 April 2020].

[28] Siubaak at GitHub, “Sval,” 2018. [Online]. Available: https://github.com/Siubaak/sval. [Accessed 29 April 2020].

[29] Lua.org, “Lua,” 2020. [Online]. Available: https://www.lua.org/. [Accessed 29 April 2020].

[30] Benoit Giannangeli, Daurnimator, Lua.org & PUC-Rio “Fengari,” 2020. [Online]. Available: https://fengari.io/. [Accessed 29 April 2020].

[31] Pierre Quentel, “Brython,” 2020. [Online]. Available: https://brython.info/. [Accessed 29 April 2020].

[32] Microsoft Corporation, “.NET,” 2020. [Online]. Available: https://dotnet.microsoft.com/. [Accessed 29 April 2020].

[33] Google LLC, “Progressive Web Apps,” 2020. [Online]. Available: https://web.dev/progressive-web-apps/. [Accessed 29 April 2020].

[34] Apple Inc., “Apple Media Service Terms and Conditions,” 2020. [Online]. Available: https://www.apple.com/legal/internet-services/itunes/us/terms.html. [Accessed 29 April 2020].

[35] Mozilla Corporation, “Add-ons in Firefox for iOS,” 2019. [Online]. Available: https://mzl.la/2duGEDd. [Accessed 18 April 2020].