ACIT4100 – Research Methods and Ethics

Current Status and Vulnerabilities of API Security

Exploring the API Security Research Field

Marius Olaussen

s336382@oslomet.no

Oslo Metropolitan University

December 9, 2019

Contents

List of Figures

List of Tables

1 Introduction

Today, *Application Programming Interfaces* – or APIs for short – are used everywhere. APIs are specifically designed to share data on networks. Today, the use of APIs has become our main channel for almost any network based server–client communication. CTO at Forum Systems, Jason Macy (2018a, p. 6), sees data as one of today's most valuable asset, and even describes data as *the new oil*.

Frequently, we read about services being hacked and personal data being leaked. Cyber security company UpGuard keeps an updated list of the biggest data breaches, ordered by people impacted (Tunggal, 2019). The Yahoo breach in late 2017, with more than 1 billion Yahoo accounts compromised, is currently still the biggest data breach of our time. However, I am shocked and disappointed by the fact that as many as *five* of the top 20 biggest data breaches, are dated as newly as 2019.

These recordings seem to correlate with the discoveries Turner (2016) made when surveying more than 100 organizations worldwide on how they manage their APIs. Several of his respondents admitted that API security implementations were either postponed to (what's left of) post-development, or, in worst case, even after the services had gone live.

1.1 My Problem

If data is one of the most valuable and sough-after asset nowadays, then why aren't APIs, which share our most valuable data across networks, *more efficiently secured*? Or, could it just be that cyber criminals constantly are ahead of us?

In my Master's thesis I am to build a modern, reliable and secure API. Thus, I want to use this text as an opportunity to learn more about the current status of the API security research field. In what way would researchers and others, interested in this research field, suggest we design and implement our APIs to ensure server–client communication with a necessary extent of data confidentiality, integrity and availability?

1.2 The Structure of this Text

During my papers selection for this text, I noticed how researchers within this research field seem divided into *two camps of researchers*. On one side, we have researchers who are enthusiastic about their own work and want to convince us that their prototypes,

models, algorithms, discoveries, or whatever, are essential contributions addressing the security problem. On the other side, we have researchers who are concerned over exposed threat surfaces, design flaws, and alike, and want to raise awareness on vulnerability implications.

Table 1 shows how evenly distributed the enthusiastic and concerned sentiments are in the papers selection. A couple of the publications stood out as somewhat more neutral. One researcher even implied there wasn't any hope (Halpin, 2018).

Table 1: The distribution of sentiments in the papers selection.

Sentiment	**Frequency**	**Frequency, in percentage**
Enthusiastic	10	45.5%
Concerned	9	40.9%
Frustrated	1	4.5%
Neutral	2	9.1%

For the sake of overview, I will structure this text by this two-sided sentimental divide. I will present and discuss the research of the *enthusiasts* in chapter 2, and the research of the *concerned* in chapter 3. Lastly, in chapter 4, I will sum up my key findings and try to draw some conclusions.

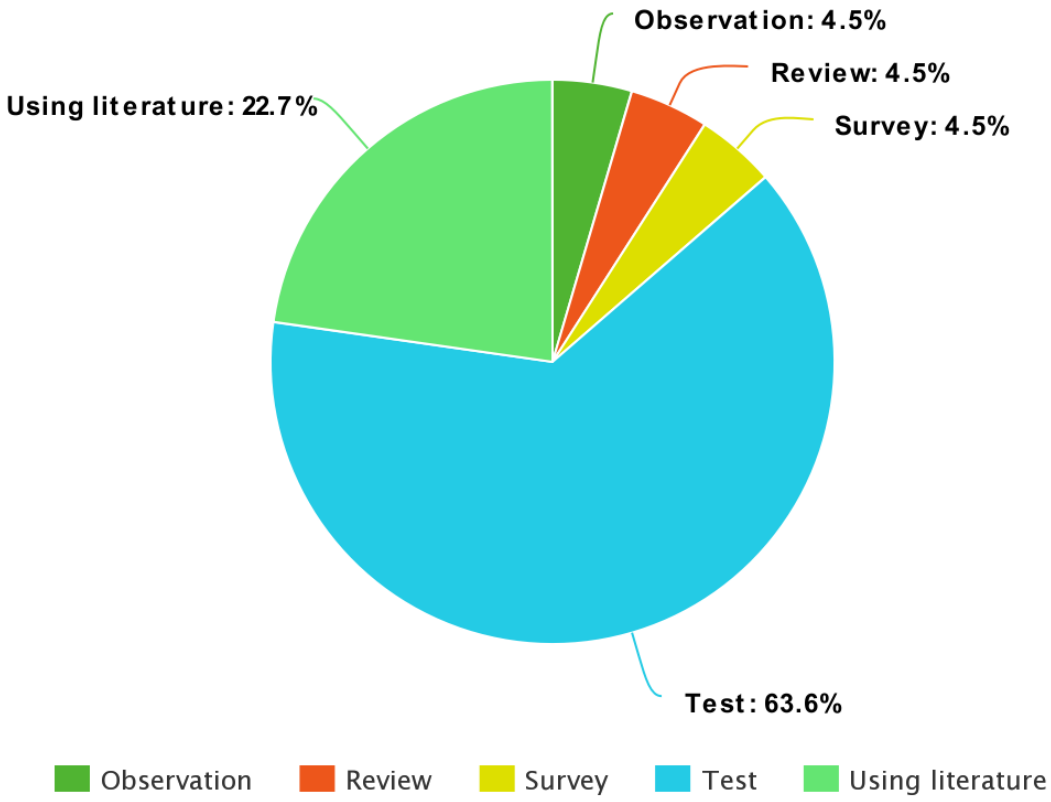
2 Visiting Research Camp A: The Enthusiasts

The half of my papers selection (45.5%) consists of publications authored by researchers, eager to present us their own work. It isn't surprising that the majority of researchers seem to be enthusiastic about their own work, at least when taking commercial aspects and personal gain into account. Academic researchers would naturally spend a lot of time on their theories and proof of concepts. Thus, if the evaluation method is solid and if the results have high scores, one should be allowed to show some enthusiasm.

The pie chart in Figure 1 visualizes how the methodologies seen in the papers selection were distributed. When categorizing methodologies, I have applied the terminology as suggested by Colin Robson (1993). This would make *tests* (63.6%), including sub methods like experiments, labs, and proof of concept, the most frequent methodology.

The second most seen methodology was *Using literature* (22.7%), often as exploratory discussions of theories, suggestions and discoveries made by other researchers. Also seen were *Observation* (4.6%), *Review* (4.6%) and *Survey* (4.6%).

Figure 1: The distribution of methodologies in the papers selection.



In the following, I will present and discuss the work of enthusiastic researchers. Due to the limitations of this text, I will include the ones I find most interesting or representative.

2.1 Neglecting the Key Security Aspects

Valvik (2012) is a stereotypical representative for the enthusiastic research camp. He's a graduate student of the University of Bergen. In his Master's Thesis he enthusiastically demonstrate how to design, build and evaluate a security API. His design is based on an existing prototype. Even though he points to several areas of improvement, he concludes that his API solution is a success, i.e. being properly secured.

If one look closer, one of his suggested future add-ons is applying HTTPS. Wouldn't the lack of encryption make this solution *useless* - at least for its intended field: eHealth?

One of the important aspects of this API was the distribution and modification of sensitive personal data. If one look even further, like in his analysis, one should notice that the main evaluation criteria is *performance*. Security isn't actually a key part of his analysis.

For obvious reasons, Valvik could surely be criticized by Research Camp B, namely the concerned researchers; *how* could one call a security API a success, when it isn't assessed correspondingly to applicable security aspects? In this case, I believe I would stand with the critics. Is this the result of neglect or a lack of effort?

2.2 Malware Classification with API Calls

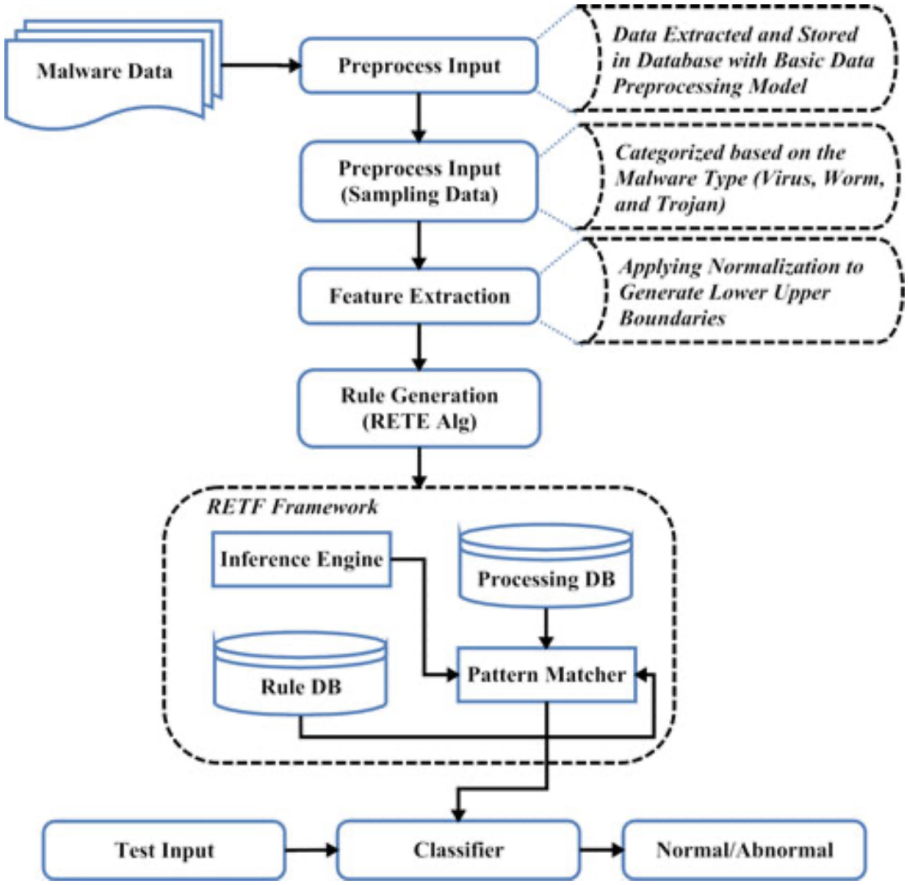
Building a malware classification system based on API calls, seems like a good idea. Two Turkish researchers, Pektaş and Acarman (2018), published some promising results last year. They provided a run-time behaviour-based classification procedure for Windows malware. They achieved this by identifying and extracting malicious patterns in API calls. Their model was trained by simulating over 17,400 malware attacks. Based on these simulations, they concluded that their model was able to classify malware attacks with a classification accuracy of 98%.

I like their method. I guess, they may be inspired by how signature- or anomaly-based intrusion detection and prevention systems, e.g. like Suricata, Bro or Snort, identify malicious network packets. Researchers from Camp B could be asking how Pektaş and Acarman intend to adapt their model to new types of malware. As we know, cyber criminals adapt to new technologies and continuously find new and more creative ways to perform attacks. Also, if the criminals are aware of how Pektaş and Acarmans model work, they could study it and make their malware even more resilient.

Jerlin and Marimuthu (2018) suggest a similar system. In comparison, their malware detection algorithms rely on machine learning techniques. They propose an API call-based malware detection system with four core components: input data normalization, upper and lower boundaries estimation, pattern matching based rules generation, and a malware classification component. Figure 2 shows how sample malware data are input to the system via the initial preprocessor. The preprocessor parses the input data and prepares it with upper and lower boundaries for rule generation. At the core of the system, the RETF framework (RE Training Framework) rely on the broadly adopted machine learning algorithm Rete to perform pattern matching and generate rules. This

system is then used to classify malwares based on API calls (p. 50).

Figure 2: The figure is taken from Jerlin and Marimuthu's (2017) publication. It shows the process flows of their Malware Detection System. The processes in the top of the figure (above the RETF Framework) constitute the preprocessor where malware data are parses, normalized and used for rule generation. The RETF Framework consists of a rule database and processors used for pattern matching and malware classification. The lower part of the figure illustrate how randomized test data are used to train the classifier. The output of the classifier is either "Normal/Abnormal".



According to their own analysis, when compared to other applicable techniques, their technique scores better. A critic of Jerlin and Marimuthu's system may point to the fact that machine learning requires both time and resources before you get reliable and usable results. Perhaps machine learning algorithms just aren't applicable for malware detection systems? As a minimum, you should at least provide sufficient training. Also, don't underestimate cyber criminals. Surely, they could also make cyber attacks relying on machine learning algorithms, too. If so, why wouldn't criminals apply *botnets* to train their systems? Just think of the unimaginable amount of zombie hosts they could use for

this. Would you ever be able to meet such an extensive amount of training by righteously provided resources? Most likely not.

Yet still, Jerlin and Marimuthu's work is worth considering. I like their approach. Their model is thoroughly assessed and the results look promising. As future add-ons, they suggest enhanced classification by clustering processes. It would be interesting to see if one could use their model to build a generic framework, which could train the malware classifier more broadly with any kind of API. That should make the rule database more precise. Also, if this system could be used within the environmental ecosystems your APIs run, as an additional detection system to Snort, Suricata, Bro or whatever you prefer, especially configured to validate API related data packets entering your network, this system could be of value for organizations relying on APIs to promote their services.

2.3 Relying on Conceptual Models

Some researchers suggest using conceptual models as means to retain your intentions throughout the process of designing, implementing and evaluating your APIs. The idea is to avoid neglecting important aspects, especially during implementation.

The work of Bond and Clulow (2006) is more than thirteen years old. Still, I think their ideas stand the test of time. They propose a conceptual framework for understanding cryptography methods. By operationalizing the four core conceptual components form, use, role, and domain, the API developer's intentions should be met in the final product. Their conceptual model is referred to as the *4AX model*. Bond and Clulow compare their model to actual security APIs to illustrate how it improves overall security. They demonstrate how their model helps the developer discover several typical security issues.

For demonstration purposes on how one could apply this model, I have included an example from Bond and Clulow (2006, p. 95), where the four core components are applied on a set of keys.

```
Key One: A PIN derivation DES key in a bank ATM network, for local
    verification of PINs, on a Thales RG7000.
Type: Keypair 14&15 (TMK/PIN)
Form: Single DES
Use: PIN Derivation
Role: Verify
Domain: KM (implicit)
```

```
Key Two: A 1024-bit private RSA signing key, for user Fred Jones in
    an applet code-signing CA, on a Chrysalis Luna CA3, accessed
    using PKCS#11
Type:
    CKA_SUBJECT = Key Two
    CKA_SIGN = TRUE
    CKA_KEY_TYPE = CKK_RSA
    CKA_MODULUS_BITS = 1024
Form: RSA, Private, 1024Bit
Use: Signature
Role: Signature Creation
```

I like the idea of breaking down design, implementation and evaluation processes into conceptual components. The 4AX model provides a simplified overview without important details being generalized and overlooked. By using standardized components one may easily combine other widely approved modeling languages like UML (Unified Modeling Language) and ORM (Object-Role Modeling). The standardization of conceptual components could most likely improve the evaluation of your UML and ORM diagrams. When designing complex entity, process and database architectures, I assume such an approach would help keeping the overview.

A critical reader may question how universally adaptable this conceptual model really is. Are form, use, role and domain sufficient for any kind of API project? Or would one need additional ones? What if a component is unidentifiable? Or, what if forms, types of use, roles or domains change dynamically over time? If this is the case, the developer may wrongfully construct a conceptual design, in which the dynamics of the components aren't identified.

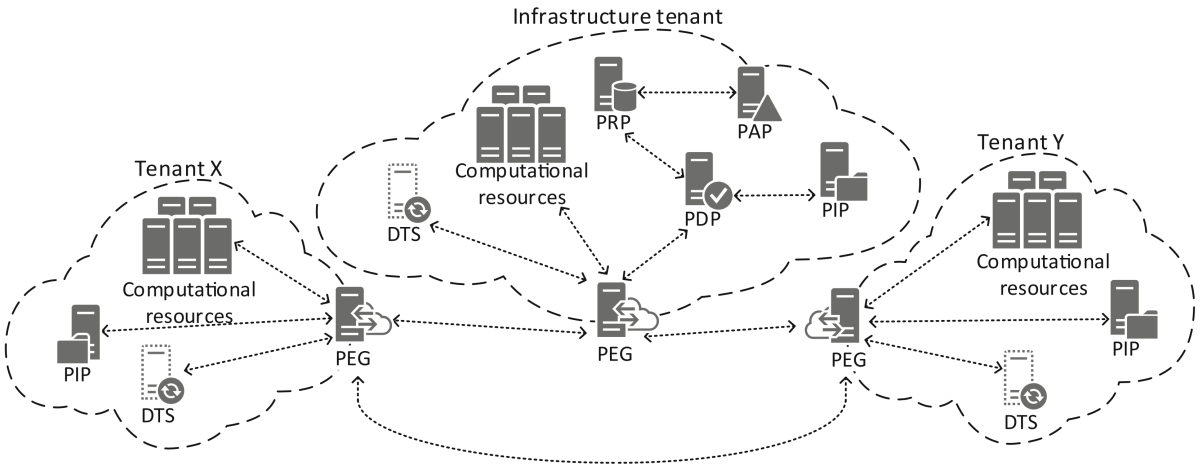
2.4 Relying on Security Policy Models

Suzic, Prünster, Ziegler, Marsalek, and Reiter (2016) propose a security policy model and an architectural framework for security governance. This architecture enables a multi-layered, context and process-aware policy enforcement in heterogeneous environments. The framework is based on derived requirements for data security and privacy in cloud federations. By having the framework based on cloud federation requirements, the model should conform to high security and privacy requirements. This should also imply the possibility of using this model on a diverse range of systems.

Figure 3 provides an overview of the architecture of their enforcement model, as proposed by Suzic et al. (2016). Notice how each of the three cloud tenants share similar or heterogeneous environmental computational resources. The figure also shows that their model apply the de-facto standard in the field of data security policy languages, *eXtensible Access Control Markup Language* (XACML), by including core XACML components, e.g. the Policy Enforcement Point (PEP), the Policy Decision Point (PDP), the Policy Administration Point (PAP), and the Policy Information Point (PIP) (Suzic, Prünster, Ziegler, Marsalek & Reiter, 2016, p. 950-1).

The relationship between these components is defined by applying PEP as a contact point for applications. PEP sends requests to the PDP as means to enforce the received decisions. The PDP contacts the PIPs to gather information, before it collects policies from the PAP. Based on these transactions, decision making and policy evaluation are done by the PDP. The result of this assessment is returned to the PEP (p. 951). If you wish to learn more about the XACML architecture, Nair (2013) provides a straightforwardly and detailed description of this stanard and its core components.

Figure 3: The figure is taken from (Suzic, Prünster, Ziegler, Marsalek & Reiter, 2016). It provides an overview of the proposed architecture. The figure shows the structure and data flow between three cloud based heterogeneous environments, referred to as tenants. Notice how the environments of each tenant share similar computational resources.



A critical reader may question what new ideas this policy model *bring to the table*, besides description a model in terms of modern cloud based terminologies. First of all, in terms of overview and means of standardization, this policy model enforces the design to describe the subjects and objects in the XACML language. This could be a desirable feature if

you already use XACML in your design.

However, there are several relevant security policy models, which describes similar architectures, to choose from already. The infamous and widely adopted model, Bell-LaPadula, provides a policy rule set where clearance labels on subjects are assigned to classification labels on objects (Harris & Maymi, 2016, p. 227). The idea of this policy model is the prevention of subjects without proper clearance accessing objects of higher classification than the subjects' clearance level. This is just one out of several models. The Biba model and the Clark-Wilson model are two other security policy models with high relevance for any security API design (p. 284).

The security policy model Suzic et al. (2016) suggests rely on access control matrices. Typically, such a matrix maps objects' access control lists (ACLs) and subjects' capability tables. The intention is to ensure proper access decisions (Harris & Maymi, 2016, p. 227). We traditionally distinguish between mandatory access control systems (MACs) and discretionary access control systems (DACs) (p. 281). Their model, just like the Bell-LaPadula model, are MAC based.

The use of a security policy model is essential to any infrastructural security framework. Since there are many to select from, one should start by comparing the features of each model to see which one is the most applicable for your infrastructure and services. I mentioned the Clark-Wilson model. This model is especially designed to protect the integrity of information. This model is based on the Biba model. Sentral terms are constrained data items (CDI), transformation procedures (TPs) and integrity verification procedures (IVPs). CDIs are data which needs to be highly protected. Only TPs are allowed to manipulate CDI data. TPs are programmed abstract operations, such as read, write and modify. The users – referred to as active agents – cannot modify CDI data directly. The users would need to be authenticated by a software, which would enable the TPs en the behalf of the users. This process is referred to as an *access triple* (user, TP and CDI). Data which the users are allowed to manipulate are referred to as unconstrained data items (UDIs) (p. 284).

This means, that if data integrity is of uttermost importance for your services, then the Clark-Wilson model is a good option. Banks and other organizations could apply this security policy model to ensure their users (active agents) would not modify the bank accounts of any other users (CDIs). Whenever they wish to perform a money transfer

from their own account to another account, they could enable this transaction from the bank's user interface (TP).

I also mentioned the Biba model (Harris & Maymi, 2016, p. 282). If your services have data from different integrity levels co-existing side by side, you could consider the Biba model. Since the Clark-Wilson model is build upon the Biba model, you should see some similarities. However, the Biba model is somewhat less complex. It really just consists of two simple rules; firstly, a subject cannot write data to an object at a higher integrity level ("no write up"). Secondly, a subject cannot read data from a lower integrity level ("no read down"). What does this mean? The intention is to preserve data integrity by keeping "clean" and "dirty" data separated. Subjects and data at a higher integrity level are protected from being corrupted by data at a lower integrity level.

Another broadly adopted security policy model is the so-called *Chinese Wall model* (Celikel Cankaya, 2011). It is heavily influenced by the Clark-Wilson model. It was first introduced in 1989 by Brewer and Nash and applied to the UK stock brokerage operations. In a context where competing stock brokers, consulted by different companies, one needed a policy which prevented conflicts of interest. The Chinese Wall model ensures no conflict of interest occurs on the same side of the wall by only granting subjects access to objects not in conflict with other objects they possess.

If one is relatively new to security policy models, I suggest start by reading the CISSP textbook by Harris & Maymi (2016). The authors provide a historical, orderly and straightforwardly overview of how such models work and a description of the most broadly adopted ones. After gaining a better understanding of how different models work, one should continue by mapping out which features are essential, important or just nice to have for your infrastructure and services. It's worth emphasizing that the design of security policy models are somewhat timeless and abstract. Because of their conceptual nature you could still see old models being used today, e.g. the military, FBI, NSA and other similar governmental organizations still use Bell-LaPadula by defining clearances to subjects and their access to classified objects, ranging from *Top Secret* to *Unclassified* (Mulligan & Elsea, 2017, p. 16).

2.5 Improved Security through Standardization

An essential feature of security designs, like when applying security policy models, is *standardization*. Standardization enables important aspects such as predictability, efficient auditing and monitoring, higher chance of identifying abnormal or malicious usage and network packets, automating security services, and more. The list is long. Standardization is a key aspect to any secure infrastructure. It is implicitly mentioned in the research projects described so far. Still, Cairns, Halpin and Steel (2016) explicitly stressed the importance of standardization, especially when regarding improving security aspects of API designs. In their publication a couple of years ago, they demonstrate how a formal modeling of protocols, such as TLS, should be applied *during* API development – and *not afterwards*. This way, the developers could not only detect attacks more easily, but also rely on APIs which are easier to understand and use.

An issue seen in several API development projects today, is the neglecting and postponing of security implementations (Turner, 2016). It is not uncommon for API projects to wait with security features until the API is in post-development, or, in worst case, even running *in production*. Cairns, Halpin and Steel's (2016) main point is that functionality and security should be implemented simultaneously. They stress the W3C Consortium to suggest a formal methodology as standardization means. They use the W3C Web Authentication API and the W3C Web Cryptography API for demonstration purposes. The result of this demonstration shows a extensive number of basic security issues, all related to the selection of algorithms (p. 134).

The W3C Web Cryptography API was exposed for three extensive cyber attacks, with one of them still present, i.e. that the usages of keys are still not preserved upon export. This bug may still be exploited in numerous ways, and may lead to not only revealing wrapped secret key material sent between a client and a server, but also disrupt authenticated key exchange (Cairns, Halpin & Steel, 2016, p. 134-5). Cairns and his fellow researchers demonstrate how easily extracted keys are attacked and retrieved from a client with a single API call. To avoid this issue when using the W3C Web Cryptography API, Cairns et al. suggest implementing functionality in your API framework, which prevents wrapping and unwrapping of keys.

Yet still, this suggestion is somewhat extreme, or at least, it implies that the W3C Web Cryptography API should be used carefully. If one is to address this issue by its

roots, one should rather address the lack of documentation. Cairns et al. believe attacks such as these could easily be avoided if the developers were guided and helped to avoid making poor design choices, e.g. implementations not meeting the required expectations for storing key material (p. 135).

I find it difficult not agreeing with Cairns et al. It is crucial that security APIs, like the W3C Web Cryptography API and the W3C Web Authentication API, are sufficiently documented and provided with examples and best practices advices to guide developers when applying these APIs in their own projects. Since security APIs, such as these two, are broadly used, the focus on usability should have high priority during the API design and development process. Myers and Stylos (2016) and others stress the importance of API usability. I will get back to their discoveries in Chapter 3.4.

Also, why would API providers even consider implementing functionality and security separately? Cairns et al. (2016) discuss issues related to applying security after the functionality implementations are done. In a worst case scenario, the algorithms selected may constitute a security breach themselves. Thus, Cairns et al. stress the importance of dealing with functionality and security at once. A plausible reason for why security implementations are sometimes postponed by API providers today, could be to ensure the projects are more cost effective, i.e. saving money and time spent. This is what Turner (2016, p. 2) referred to as *API economy*. For the majority of users, the lack of functionality is *more visible* than the lack of security – at least, as long they don't end up as victims of a cyber attack.

I will continue this discussion in Chapter 3.2. For now, just keep in mind that, in terms of API economy, it could very well be that attacks not directly harming the organizations themselves, but "only" their clients, *may be overlooked* as long as the attack doesn't result in the organization loosing money and reputation. Remember, security implementations cost money and time. If the total amount of money required to harden the attack surface is higher than the value of customers, reputation or data lost, why should we expect an organization to allocate their manpower, money and time on the matters?

2.6 Chapter Summary

Table 2 below is provided to give a brief summary of the publications presented and discussed in this chapter. For the sake of overview, I have tried to make the description

as brief as possible. For the same reason, I will only recapitulate the key methodologies and conclusions. I have, mostly, left out my personal views and other aspects of my discussion. For reference and verification purposes, I have included the publication IDs. See the appendix for details.

Table 2: Summary of the publications discussed in Chapter 2. IDs are included for reference and verification purposes. See appendix for more details.

Author(s)	**Description**	**IDs**
Valvik	Designs, builds and evaluates a security API, based on an existing prototype. Fails to assess security properly.	1
Pektaş, Acarman	Provide a malware classification system based on API calls. Trained on over 17,400 simulation attacks.	22
Jerlin, Marimuthu	Suggest a API call-based malware detection system with 4 core components: input data normalization, upper and lower boundaries estimation, pattern matching based rules generation, and a malware classification component.	7
Bond, Clulow	Propose a conceptual framework to help understand cryptographic methods, as means to maintain the developer's intentions during the development process. Four core conceptual components: form, use, role, domain.	13
Suzic, Prünster, Ziegler, Marsalek, Reiter	Propose a security governance architecture to enable a multi-layered, context and process-aware policy enforcement in heterogeneous environments. In short, yet another security policy model, with similarities to models such as Bell-LaPadula, Biba and Clark-Wilson.	19
Cairns, Halpin, Steel	Use the W3C Web Authentication API to demonstrate the need of a formal methodology as standardization means to enforce security implementation during API development.	15

3 Visiting Research Camp B: The Concerned

As mentioned in Chapter 1.2, the papers selection seems to be divided into *two camps of researchers*, sentiment-wise. Until now, I have presented and discussed projects from the first research camp. In the following, I will present and discuss projects from the second research camp – namely, the *concerned researchers*. What these researchers share in common, are their concern over exposed threat surfaces, design flaws, bugs and alike, and their focus on raising awareness on vulnerability implications.

3.1 API Security is a Disjointed Affair

Three years ago, the North-American cyber security company, Distil Networks, asked the technology research and marketing company, Ovum Consulting, to conduct a qualitative survey on API management. Rik Turner (2016), a senior analyst for infrastructure solutions at Ovum Consulting, was leading this study. As the subtitle of the survey report implies, API security was the key focus area: *CIOs/CISOs need to know how API security is managed within their own organizations.*

Before I continue, I will give a brief description of the leadership roles CIO and CISO, since these may be new to some of us. They were first introduced in the mid 90s. The first person hired as a CISO, was Stephen Katz. The company who was hiring, was the North-American bank Citigroup (Fazzini, 2018). CIO is short for *Chief Information Officer*. The CIO is usually the person leading the information technology within an organization, at executive board level (Beal, 2019). CISO is short for *Chief Information Security Officer*. The CISO is the executive responsible for an organization's information and data security, at executive board level (Fruhlinger, 2019).

In Turner's (2016, p. 2) survey, more than 100 organizations in North America, Europe, and Asia-Pacific, representing different industries and market sizes, were asked about how they manage their APIs. The questions range from general API usage to adoption of API management platforms, and applied security features. The overall goal was to gain understanding of the CIOs' and CISOs' general awareness and concerns regarding security in a cost-benefit perspective, also referred to as *API economy* (p. 2).

Figure 4: The figure is taken from the survey report. It shows the results when the CIOs and CISOs were asked what the purpose of their APIs was.

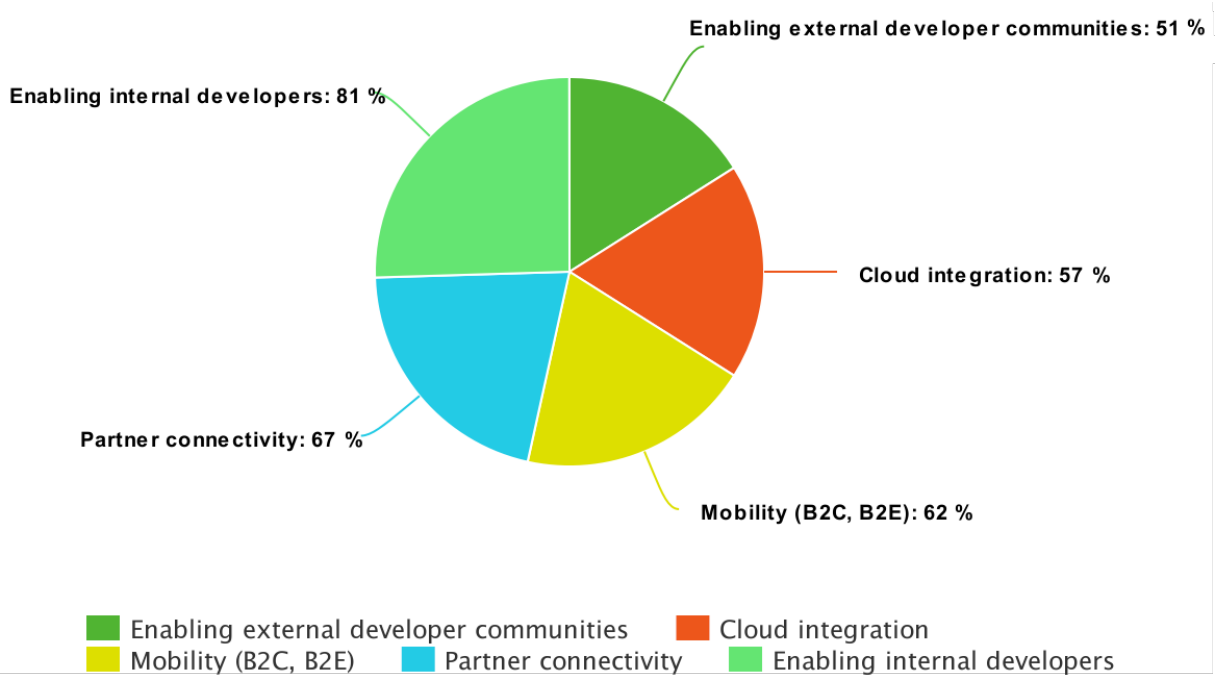
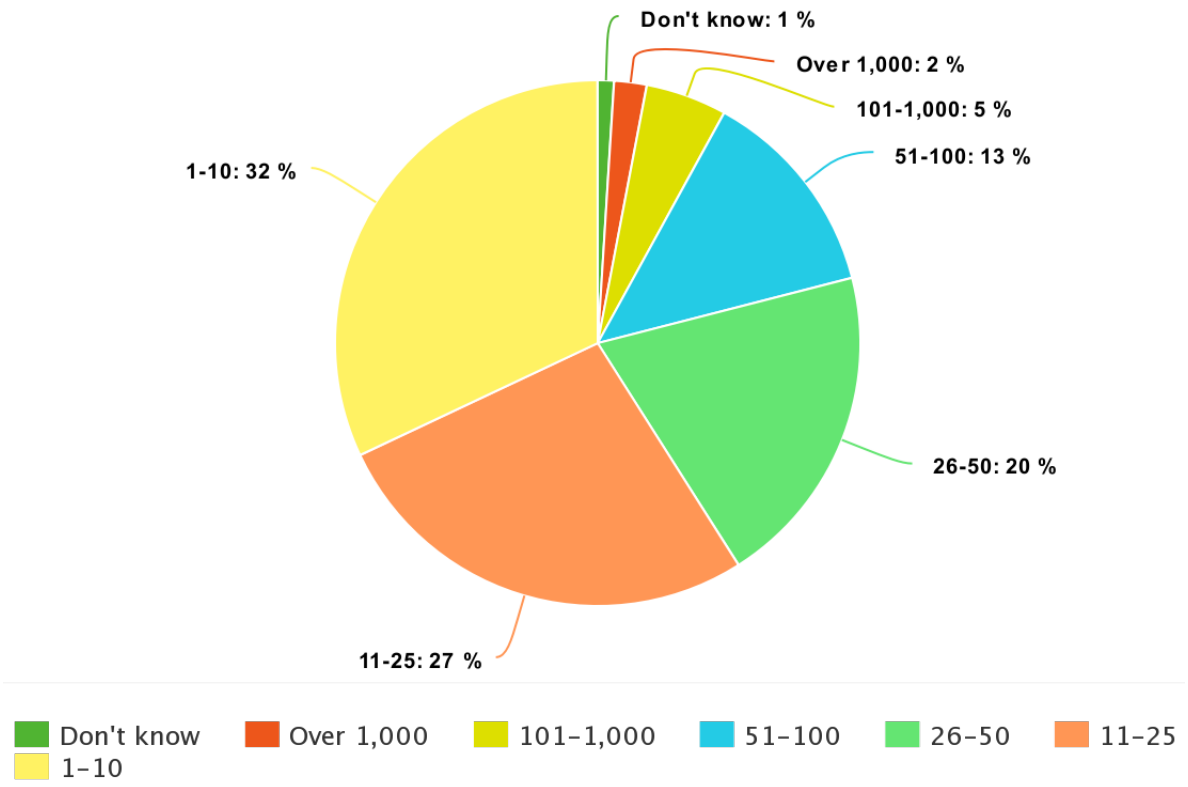


Figure 4 shows how the CIOs and CISOs replied, when asked what the purpose of their APIs was. 81% used APIs to enable their internal developers easier access to company data and resources, and 51% to enable external developer communties. Other purposes mentioned were cloud integration (57%), partner connectivity (67%), and mobility – either B2C or B2E (62%).

In general, the CIOs and CISOs said they are concerned about API security. Yet still, 17% said they were not concerned with API security *at all*. As Turner suggests, a reason for this lack of concern could be a result of this group of respondents having their APIs taken care of by commercially supported API management platforms.

A majority of the organizations writes most of their APIs *in-house*. Figure 5 shows what the CIOs and CISOs replied, when asked how many APIs their organization maintain, build, or publish. Over 40% of the respondents are running more than 25 APIs, and 7% more than 100 APIs (Turner, 2016, p. 3).

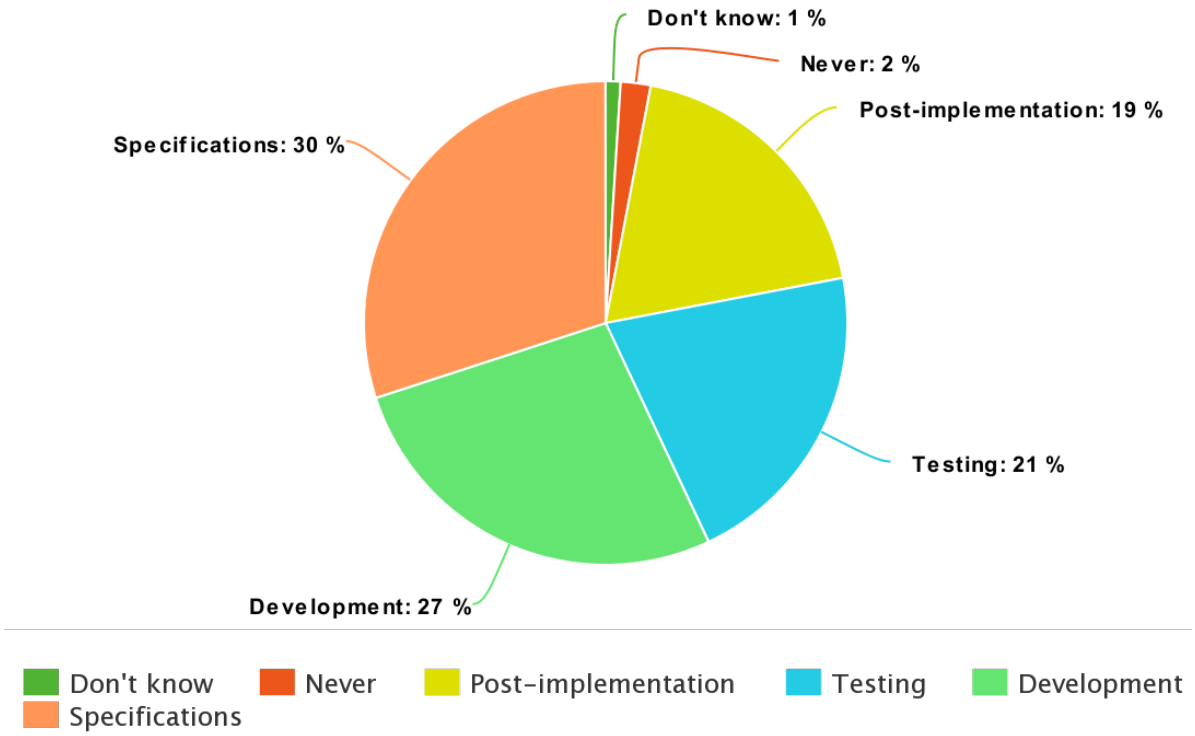
Figure 5: The figure is taken from the survey report. It shows the results when the CIOs and CISOs were asked how many APIs their organization maintained, built, or published.



Yet still, it is shocking to discover that nearly one-third of APIs end up going through the requirements specification, without even being looked at by the in-house IT security teams. This is shown in Figure 6 on the next page. This figure also show us how nearly 30% of APIs pass the development stage without the IT security teams even providing thoughts or comments about their current implementation status. As many as 21% of APIs do not get any input from the IT security teams, whatsoever, before they *go live* (p. 2). Shocking.

The 1% of the CIOs and CISOs saying they don't know at what point they engage their IT security team, when building APIs, could be a result of them having the API management outsourced to external and commercial platforms. Also, notice that 2% even said they never engage any security team. It's somewhat vague in Turner's report, but this could be organizations who let their *developers* manage security implementations, meaning that they *do not have* exclusive security teams, only developers. If this is the case, this is a methodological weakness in Turner's survey. However, I do not know whether this may be the case or not.

Figure 6: The figure is taken from Turner's survey (2016). It shows the results when the CIOs and CISOs were asked at what point they engage their IT security team, when they are building out APIs.



One of Turner's (2016, p. 10) main conclusions was that security implementations have an overall surprisingly *low priority*. The results from his survey seem to draw a picture of CIOs and CISOs not fully understanding how API security is to be managed within their own organizations. When asked what protection the organizations' current API management provided, only 21.9% had protection across all specified attack vectors: malicious usage, developer errors, automated scraping, web and mobile API hijacking. In the worst case scenario, 8.3% of the respondents said they had no protection related to these four attack vectors. When failing to ensure a required minimum of security implementations, the APIs may increase the organization's attack surface, exposing application structure and data to potential hackers (p. 2).

Some of the respondents said they had their APIs protected from malicious usage or automated scraping by *a bot*. Whenever this bot suggested there was an attack attempt, it would pull down the online content and data within minutes. This is not a good idea. If one consider that more hackers now begin to target APIs directly, such an security implementation would make an enterprise a victim of DoS attacks (p. 2).

To improve the efficiency of security implementations, Turner (2016, p. 2) suggests the API providers collaborating and dealing with security aspects, *collectively*. I consider Turner's study to be valuable. It reveals useful insights on how organizations respect API security implementations. Their postponing of security implementations imply a necessity of always assessing the security aspects of a public API, before using them in your own development. Do not blindly expect the public APIs to be properly secured. Explicitly ensure they provide sufficient extent of data confidentiality, integrity and availability. In regard of *API economy*, I will continue this discussion in Chapter 3.2 below.

3.2 Public APIs Could Leak Personal Data

An article which demonstrates the importance of assessing public APIs before usage, is the article Bhuiyan et al. (2018) published in the International Journal of Engineering and Technology, last year. In their article, they tried to map out several known security risks and vulnerabilities related to some well-known public APIs. They demonstrate how the current status and dependencies of these public APIs constitute *several security issues*. They provide proof of concept to their claim, and want to raise awareness among developers using public APIs in their development. Their demonstration provides evidence on how confidential personal data are exposed and *leaked via public APIs*. Many of the vulnerabilities the researchers identified, are highly alarming and mostly related to lack of proper *user authorization* when co-existing with other API frameworks.

The work of Bhuiyan et al. (2018) illustrates the need to improve the security aspects of APIs we rely on in our daily online activities. Their work is important and correlate to what Turner (2016) found in his survey; a developer cannot expect a proper extent of security implementations in public APIs. Also, note that this article was published last year. We should expect an unchanged status in 2019.

The same year, Jason Macy (2018a) wrote an article in the magazine *Network Security*, titled *API Security: Whose Job is it Anyway?* It's a good question. Since Turner's (2016) survey draw a picture of organizations tending to postpone or skipping a required extent of security implementations, someone should ask such a question. Could it be that the company CIOs and CISOs do not want to allocate their resources for security implementations – if any security breach and data loss wouldn't *directly* harm the company? I mean, take a bank, for instance. If one of their customer becomes a victim of identity

fraud, e.g. as a result of an eavesdropping or spoofing attack, could we expect that this bank would allocate resources – staff and other expenses – to strengthen their APIs, in such a way that similar attacks would not occur again?

A bank is a commercial institute. In a worst case scenario, a bank could postpone these expenses until they would *have to* take actions. I mean, if the fraud occurrence harms the customer more than the bank, i.e. that the bank would not loose their reputation nor experience any kind of harm, it could be likely that the bank decides to postpone any security enhancements.

Remember when Google decided to block non-encrypted websites back in 2015, to ensure their users safety online? The Norwegian bank organization Nordea was affected by this shutdown (Bakken, 2015). Any websites relying on obsolete encryption ciphers, like RC4 in Nordea's case, would not be accessible via the Google Chrome web browser. Nordea got a grade F and failed hard, when a Norwegian developer at Opera ran all Norwegian banks' online banking services through Qualys SSL Labs' TLS test (Aleksandersen, 2015). While SpareBank1, Storebrand, and Skandiabanken got grade A, Gjensidige, KLP, and Nordea got grade F. Nordea was vulnerable to POODLE attack, had weak SHA-1 certificate chain, and didn't ensure forward secrecy, nor secure renegotiation in their online banking services.

Even if Nordea replaced the obsolete encryption cipher, they didn't enable TLS for the entire website – only their banking services (nettbanken). Thus, cyber criminals could potentially redirect their customers from the main page to a phishing site. If you weren't careful, you could end up typing your credentials at phising sites, e.g. like https://nordea.nettbank.log.in.

Today, Nordea has properly encrypted their online services. Yet still, we see headlines saying Nordea sends their customers PIN codes by postal mail (Pedersen, 2018). What does this tell us? Personally, I would use headlines like these as guidelines, or implications on what to expect of them, when regarding their focus on security implementations. If organizations postpone security matters, most likely due to economical reasons, you should be careful when using their services. Maybe one should follow Google's example and avoid organizations who do not respect their customers safety to a necessary extent (Bakken, 2015)? The same goes for postponing security implementations in API frameworks due to the organizations' API economy considerations.

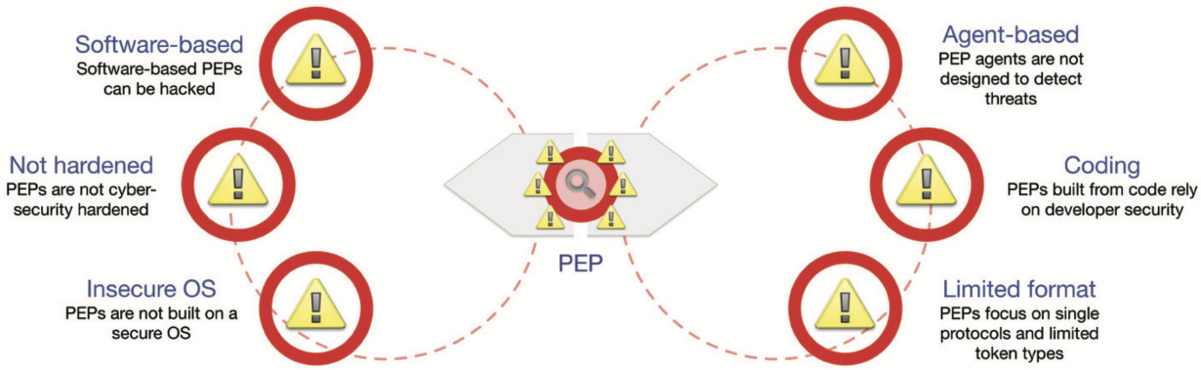
3.3 Security Implications of the API Ecosystems

Macy (2018a) wants to raise awareness on API technology weaknesses, as compiled by OWASP. He points to the OWASP Top 10 list of web application risks as an expected minimum for any network based application. As an extension to this list, he stresses that security specialists need to shift their focus towards *API security products*, rather than API solutions. It is somewhat vaguely explained, but his point is that the *baseline matters*. This is an interesting observation, which I have not seen mentioned explicitly in other publications. It's actually rather obvious. It really doesn't matter how well secured an API is, if the environments it rely on, are poorly secured.

In his article, Macy (2018a) strikes hard. Many technologies and models, as suggested by other researchers, are described with few superlatives. Remember the security policy enforcement model, suggested by Suzic et al. (2016)? Macy (2018a, p. 9) has a more concerned approach to cyber-secure policy enforcement points (PEPs). He believes they may, by their very nature, constitute a vulnerability to your infrastructure. This doesn't mean he wouldn't suggest applying them in your model. He just wants to raise awareness about the fact that having more PEP entries would be like adding more doors to your house. Then, it doesn't matter how secured your front door is, if one of your back doors are wide open. He shares this view with other security aware activists (Bot, 2018).

Figure 7 is taken from his article. It illustrates the fact that, by how PEPs are designed, they are not security hardened. When software-based, they may be hacked. When agent-based, they aren't designed to detect threats. When built by an in-house developer, they aren't more secure than the developer makes them. Neither are they built on a secure operating system. Also, because of their focus on single protocols and limited token types, PEPs also represents vulnerabilities regarding their format, by design. This means you should use PEPs carefully.

Figure 7: The figure is taken from Macy (2018a). Macy provided this figure in this article to illustrate his claim that PEPs make your infrastructure vulnerable to cyber attacks.



The same year, Macy (2018b) published a second article. In this publication he clarifies how the baseline should be secured: by building *a secure API gateway*. If you apply proactive checks, you should be able to avoid compromising your organization's IT infrastructure. Macy suggests proactive monitoring of data communication between the servers and their clients as means to avoid exposing your data through applied APIs.

In addition to proper monitoring of any network based data communication, one should properly lock down the operating systems the APIs are running on. *Monitoring* and intrusion detection systems are key features in your infrastructure. Macy's point is that organizations would need to identify and block malicious attacks on their *entire* exposed threat surface, not only the APIs themselves. Macy makes good points. Why haven't other researchers mentioned something similar?

3.4 Improving API Usability

Another important security aspect is usability. Myers and Stylos (2016) want to raise awareness on user-focused API development as means of improving overall usability and security. Their point is that APIs are often improperly implemented. This could potentially result in bugs and, in worst case scenario, significant security issues. This is a huge problem. Modern network based applications tend to make an extensive use of public APIs. Myers and Styles suggest that the providers improve API usability and security, by evaluating and designing APIs with their users in mind.

I agree that usability is an important aspect of API security. Myers and Stylos' suggestions seem to some extent correlate with what Bhuiyan et al. (2018) demonstrated.

Still, Bhuiyan and his fellow researchers raise awareness of API vulnerabilities as a result of poor API design and implementation, while Myers and Stylos stress improper adoption of public APIs resulting in security issues. There's a difference. Yet, there are similarities as well.

Myers and Stylos make a good point. When designing APIs, one should always try to keep the intended users in mind. Ideally, API functionality should be intuitive and unequivocally. Documentation should be provided as well as examples of how to perform API calls. This is essential to help developers avoid erroneous implementations.

There are several researchers within the API security field who are focused on API usability. Another interesting publication is the article Wijayarathna and Arachchilage (2019) published this year. In this article they describe a case study they conducted. The method applied was an experiment test lab where 11 developers were observed during a period of two hours. The developers were to build an application and secure it with the Java Secure Socket Extension API (JSSE). They identified 59 usability issues, which they categorized into 15 cognitive dimensions.

Their analysis provided important insights on how TLS APIs and similar security APIs should be improved with the users in mind. The security flaws identified in their observations could likely imply a correlation between API usability and how the security APIs are designed, implemented and documented – at least when it comes to the JSSE API. Wijayarathna and Arachchilage stress the importance of improving the usability, especially when regarding security APIs. Correct implementations of security APIs are crucial to ensure developers are building properly secured network based applications.

3.5 A Cost-Benefit Approach

Another important aspect of API security is related to the idea of disabling unused web browser functionality. Snyder, Taylor and Kanich (2017) gave an interesting speech regarding these matters on the ACM SIGSAC Conference on Computer and Communications Security in 2017. They suggest a cost-benefit based methodology by avoiding web services *default access to browser functionality*. As clients have nearly no direct requirements, there really are no need of allowing default access to an extensive part of the web APIs. Such an access would increase the threat surface unnecessarily.

Table 3: These numbers are taken from (Snyder, Taylor & Kanich, 2017). They show the cost and benefit statistics for the evaluated conservative and aggressive browser configurations, as suggested by Snyder and his fellow researchers. The # character is used to inform that they numbers in the corresponding row are shown by frequency (N).

Statistic	**Conservative**	**Aggressive**
Standards blocked	15	45
Previous CVEs #	89	123
Previous CVEs %	52.0%	71.9%
LOC Removed #	37,848	53,518
LOC Removed %	50.00%	70.76%
% Popular sites broken	7.14%	15.71%
% Less popular sites broken	3.87%	11.61%

As proof of concept to their claim, they applied their cost-benefit based methodology on 74 web APIs in several modern browsers. The cost and benefit statistics for the evaluated conservative and aggressive browser configurations are provided in Table 3. As we see, their analysis shows that 15 of 74 APIs could easily remove as much as 50 % of their current access to web browser functionality, without the API functionality being affected at all. This work implies the importance of bypassing or locking any non-required functionality down.

It is already of interest to discover that only approximately 15% of the popular sites in their evaluation breaks when an aggressive browser configuration is applied. By aggressive configuration, they mean blocking APIs like Google's File API standard, the Media Source Extensions standard, which services like YouTube rely on, and similar APIs. This is an interesting read. Since it may imply we should, by default, apply a rather aggressive web browser configuration. Then, whenever the sites and services we use require additional access, we could allow this temporarily.

These thoughts correlate closely with the API security gateway model, as described by Macy (2018b). In general, the threat surface should be as small as possible. As a matter of fact, Snyder et al. (2017) showed us the kind gesture of writing a web browser extension for all major browsers, which would help us, the users, to adapt our browser configurations to the currently active and non-active APIs.

It would also be an interesting project to explore the possibilities of building a *wrapper API framework*, which could be applied to any network based services. This wrapper API framework could function similar to the extension Snyder et al. (2017) built, by ensuring aggressive configurations. This may help the developers block unwanted access attempts, which they overlooked at the time of designing their APIs. This could improve the overall security, when building APIs. Naturally, one would need to start by verifying whether this is possible or not. In worst case scenario, an aggressive wrapper API framework could lead to frustrating developers and APIs not functioning as intended, e.g. by blocking non-malicious or abnormal usage (false positives).

3.6 Chapter Summary

Similar to the summary for Chapter 2, I will provide a summary for Chapter 3. Table 4 is provided to give a brief summary of the publications presented and discussed in this chapter. Like in Chapter 2, I will only recapitulate the key methodologies and conclusions. I have, mostly, left out my personal views and other aspects of my discussion. For reference and verification purposes, I have included the publication IDs. See the appendix for details.

Table 4: Brief summary of the publications authored by concerned researchers. IDs are included for reference and verification purposes. See appendix for more details.

Author(s)	**Description**	**IDs**
Turner	Conducts a qualitative survey on API management for 100+ organizations worldwide. Found among other things that security implementations are postponed to latter part of development or even after the services have gone live.	3
Bhuiyan, Begum, Rahman, Hadid	Tests a set of public APIs as proof of concept to their claim that several public APIs constitute securtity risks. Data are leaked, especially due to lack of authorization, when public APIs co-exist with your API framework.	10

Continued on next page

Author(s)	Description	IDs
Macy	Discusses API technology weaknesses, as compiled by OWASP Top 10 Web Application Risks. Wants security specialists to focus more on API solutions, rather than products, because the baseline matters.	2
Macy	Provides implementation recommendations on how to build a secure API gateway, e.g. by applying proactive checks and monitoring business transaction. In short, raises awareness on how the entire environment your APIs run in, would also be a part of the threat surface.	9
Myers, Stylos	Raise awareness on user-focused API development, as means of improving overall usability and security. Want developers to improve usability by designing APIs with the users in mind.	12
Wijayarathna, Arachchilage	Evaluate the usability of the Java Secure Socket Extension API (JSSE). Gave 11 programmers 2 hours to build a secure web application. Found 59 usability issues and discuss how these relate to poor usability design.	11
Snyder, Taylor, Kanich	Suggests a cost-benefit based methodology of giving websites default access to browser functionality. Demonstrate on 74 web APIs in modern browsers. Found that 15 of these would reduce the threat surface by 50% by locking down access. And this didn't affect the API functionality at all.	18

4 Summary and Conclusions

I have now presented and discussed a couple of research projects, representing both research camps. In the following, I will try to highlight some key aspects of what I have learnt from this exercise. Finally, I will try to draw some conclusions based on these. But firstly, I should provide a recapitulation of my problem statement, for the sake of overview.

4.1 Recapitulation of my Problem Statement

In this text, I have presented and discussed publications which provide valuable and thoughtful insights to the API security research field. As I am to build a modern, reliable and secure API in my Master's Thesis, I wanted to use this text as an opportunity to learn more about the current status of this research field. The question I asked in my problem statement was as follows:

> *In what way would researchers and others, interested in this research field, suggest we design and implement our APIs to ensure server–client communication with a necessary extent of data confidentiality, integrity and availability?*

4.2 Both Research Camps Play Their Role

While some researchers focus on providing means to meet the security problem in general or certain parts of it, other researchers concentrated on raising awareness on the vulnerabilities and flaws in related technologies, methodologies and designs. My investigation has showed me several things. First and foremost, I understand that *both* research camps play their role in pushing the field of API security forward. The enthusiasts don't just point to the problems. They provide solutions to them as well. Without these researchers, we would need to come up with the prototypes, models, algorithms, policies and other security concepts ourselves. The concerned researchers also play their part. These researchers aren't convinced by superlatives and buzzwords. They demand facts, proof of concept and unbiased evaluation results.

Obviously, 22 publications aren't enough to draw the whole picture. However, these publications should at least help us notice some of the current trends within this research field. As we have seen, these publications range from graduate students thrilled about having a functioning solution, to building advanced malware detection systems, which are trained by identifying malicious API usage. By the way, training a malware classification system by feeding it with sequences of API calls, would not only help us better understand how cyber criminals identify and misuse bugs and security issues in APIs for their own gain, but also provide insights on how to reduce the attack surface our APIs represent.

4.3 Important Security Aspects Rely On Standardization

Conceptual models are valuable. By abstracting and breaking down complex architectures and data processes into conceptual components, developers have better opportunities of retaining a necessary extent of overview throughout the development process. By designing API frameworks in terms of conceptual components, e.g. like form, use, role, and domain, as suggested in the 4AX model (Bond & Clulow, 2006), conceptuals models also allow *standardization*. Standardization enables important security aspects, such as predictability, efficient auditing and monitoring, higher chance of identifying abnormal or malicious usage and network packets, automating security services, and more.

Security policy models enforce standardization. I have presented the policy enforcement model, as suggested by Suzic et al. (2016). However, since there are several models to choose from, I suggest starting by gaining better insights on how the most broadly adopted ones, like Bell-LaPadula, Biba, and Clark-Wilson, work. When selecting a model, you should try mapping out their features and compare them to which features are essential to your infrastructure and services.

4.4 The Baseline Matters

As the short amount of papers in my selection implies, there aren't any single solution to the security problem. One could ask if it's even possible to build a properly secured API. Macy (2018a) would agree to this. He wanted to raise awareness of important *the baseline* is. It doesn't matter how well secured your API is, if the environment it runs on is poorly secured. The operating system needs to be locked down. All business transactions going in and out of your network, would need proper monitoring and auditing.

Also, Snyder et al. (2017) gave us something to contemplate. Why would we allow APIs default access to a large extent of the functionality in the web browsers we use daily? They demonstrate how hardening the browser configuration really doesn't make websites crash. Even when enabling an aggressive configuration, only 15% of the popular sites broke. This implies we might allow APIs too broad access via our web browsers. Why allowing default functionality, when a large part of it isn't required? Also, remember that public APIs accessing your browser might leak sensitive data (Bhuiyan et al., 2018).

4.5 API Economy

API economy also plays its part. Building APIs isn't cheap. The survey Turner (2016) conducted, illustrates how CIOs and CISOs tend to postpone security implementations. Despite the fact that APIs have become our main channel for distributing our data, both personal and commercial, online, Turner draws a picture of CIOs and CISOs not fully understanding how API security is to be managed within their own organizations. This implies that we should never blindly presume that public APIs are properly secured. Bhuiyan et al. (2018) proved this when they demonstrated how API projects may leak sensitive data, when co-existing with public APIs.

4.6 Security APIs Should Be Intuitive and Unequivocally

Usability issues could also lead to vulnerabilities and security flaws. Myers and Stylos (2016) demonstrated how lesser intuitive security APIs with poor documentation could result in the developers misinterpret and implement security APIs erroneously. Improper implementation may increase the attack surface of an organization's infrastructure and resources. Thus, security API providers should design APIs with the users in mind.

4.7 Conclusions

As I have learnt, this research field is *constantly evolving*. Not only because of new technologies and ideas, but also because of the fact that cyber criminals adopt new technologies and ideas to their attacking systems, too. There might never be any ultimate and timeless answer to the security problem. The battle between the cyber criminals and the API providers will continue to rage on, restlessly.

Today, we see highly sophisticated, dynamic and even automated attacks. If we are to stand a chance, we need to stand together, *collaborate* and keep on fighting, as Turner (2016) suggested. Securing our APIs and the ecosystems they live in, is a never-ending job. Also, CIOs and CISOs need to fully understand how they should manage API security in their organizations. Security implementations need to be dealt with *simultaneously* as functionality implementations. APIs carry our most valuable assets – *our data*. API providers should need to start acting accordingly.

References

Aafer, Y., Tao, G., Huang, J., Zhang, X. & Li, N. (2018). Precise android api protection mapping derivation and reasoning. *CCS '18 Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 1151–1164. doi:10.1145/3243734.3243842

Aleksandersen, D. (2015, April 29). Gtls ratings for norwegian banks. Retrieved November 29, 2019, from https://www.ctrl.blog/entry/tls-ratings-norwegian-banks.html

Bakken, J. B. (2015, December 2). Googles mest avanserte nettleser blokkerer nå all trafikk til nordeas nettbank: Av hensyn til brukerens sikkerhet. Retrieved November 29, 2019, from https://www.dn.no/nordea/it-sikkerhet/hacking/nordea-blokkeres-pa-grunn-av-darlig-sikkerhet/1-1-5519854

Beal, V. (2019, November 29). Cio. Retrieved November 29, 2019, from https://www.webopedia.com/TERM/C/CIO.html

Beer, M. I. & Hassan, M. F. (2018). Adaptive security architecture for protecting restful web services in enterprise computing environment. *Service Oriented Computing and Applications*, *12*(2), 111–121. doi:10.1007/s11761-017-0221-1

Bhuiyan, T., Begum, A., Rahman, S. & Hadid, I. (2018). Api vulnerabilities: Current status and dependencies. *International Journal of Engineering and Technology(UAE), 2018, Vol.7(2), Part 3*, *7*(2), 9–13. doi:10.14419/ijet.v7i2.3.9957

Bond, M. & Clulow, J. (2006). Integrity of intention (a theory of types for security apis). *Information Security Technical Report*, *11*(2), 93–99. doi:https://doi.org/10.1016/j.istr.2006.03.003

Bot, H. (2018, February 22). Api policies - pap, pdp, pep and pip. oh my! Retrieved November 29, 2019, from https://www.yenlo.com/blog/api-policies-pap-pdp-pep-and-pip.-oh-my

Cairns, K., Halpin, H. & Steel, G. (2016). Security analysis of the w3c web cryptography api. In L. Chen, D. McGrew & C. Mitchell (Eds.), *Security standardisation research* (pp. 112–140). Cham: Springer International Publishing.

Celikel Cankaya, E. (2011). Chinese wall model. In H. C. A. van Tilborg & S. Jajodia (Eds.), *Encyclopedia of cryptography and security* (pp. 203–205). doi:10.1007/978-1-4419-5906-5_777

Cortier, V. & Steel, G. (2014). A generic security api for symmetric key management on cryptographic devices. *Information and Computation*, *238*, 208–232. Special Issue on Security and Rewriting Techniques. doi:https://doi.org/10.1016/j.ic.2014.07.010

Fazzini, K. (2018, July 21). Here's what cybersecurity professionals at companies actually do, and why they're so vital. Retrieved November 29, 2019, from https://www.cnbc.com/2018/07/20/what-is-ciso-chief-information-security-officer.html

Fruhlinger, J. (2019, November 29). What is a ciso? Responsibilities and requirements for this vital leadership role. Retrieved November 29, 2019, from https://www.csoonline.com/article/3332026/what-is-a-ciso-responsibilities-and-requirements-for-this-vital-leadership-role.html

Halpin, H. (2018). A roadmap for high assurance cryptography. In A. Imine, J. M. Fernandez, J.-Y. Marion, L. Logrippo & J. Garcia-Alfaro (Eds.), *Foundations and practice of security* (pp. 83–91). Cham: Springer International Publishing.

Harris, S. & Maymi, F. (2016). *Cissp all-in-one exam guide, seventh edition* (7th). McGraw-Hill Education Group.

Islam, C., Babar, M. A. & Nepal, S. (2019). A multi-vocal review of security orchestration. *ACM Comput. Surv. 52*(2), 37:1–37:45. doi:10.1145/3305268

Jerlin, M. A. & Marimuthu, K. (2018). A new malware detection system using machine learning techniques for api call sequences. *Journal of Applied Security Research*, *13*(1), 45–62. doi:10.1080/19361610.2018.1387734. eprint: https://doi.org/10.1080/19361610.2018.1387734

Macy, J. (2018a, September 19). Api security: Whose job is it anyway? Retrieved September 16, 2019, from https://www-sciencedirect-com.ezproxy.hioa.no/science/article/pii/S1353485818300886

Macy, J. (2018b). How to build a secure api gateway. *Network Security*, *2018*(6), 12–14. doi:https://doi.org/10.1016/S1353-4858(18)30056-4

Mulligan, S. P. & Elsea, J. K. (2017, March 7). Criminal prohibitions on leaks and other disclosures of classified defense information. Retrieved November 29, 2019, from https://fas.org/sgp/crs/secrecy/R41404.pdf

Myers, B. A. & Stylos, J. (2016). Improving api usability. *Commun. ACM*, *59*(6), 62–69. doi:10.1145/2896587

Nair, S. (2013, November 19). Xacml reference architecture. Retrieved November 29, 2019, from https://www.axiomatics.com/blog/xacml-reference-architecture/

Nguyen, H. V., Tolsdorf, J. & Lo Iacono, L. (2017). On the security expressiveness of rest-based api definition languages. In J. Lopez, S. Fischer-Hübner & C. Lambrinoudakis (Eds.), *Trust, privacy and security in digital business* (pp. 215–231). Cham: Springer International Publishing.

Padmanaban, R., Thirumaran, M., Anitha, P. & Moshika, A. (2018). Computability evaluation of restful api using primitive recursive function. *Journal of King Saud University - Computer and Information Sciences*. doi:https://doi.org/10.1016/j.jksuci.2018.11.014

Pedersen, L. H. (2018, January 5). Storbank sender fortsatt alle pinkoder i posten. Retrieved September 16, 2019, from https://www.nrk.no/ostfold/nordea-folger-ikke-bransjens-anbefaling-1.13852791

Pektaş, A. & Acarman, T. (2018). Malware classification based on api calls and behaviour analysis. *IET Information Security*, *12*(2), 107–117. doi:10.1049/iet-ifs.2017.0430

Robson, C. (1993). *Real world research: A resource for social scientists and practitioner-researchers*. Blackwell, Oxford.

Snyder, P., Taylor, C. & Kanich, C. (2017). Most websites don't need to vibrate: A cost-benefit approach to improving browser security. In *Proceedings of the 2017 acm sigsac conference on computer and communications security* (pp. 179–194). CCS '17. doi:10.1145/3133956.3133966

Srivastava, V., Bond, M. D., McKinley, K. S. & Shmatikov, V. (2011). A security policy oracle: Detecting security holes using multiple api implementations. In *Proceedings of the 32nd acm sigplan conference on programming language design and implementation* (pp. 343–354). PLDI '11. doi:10.1145/1993498.1993539

Suzic, B., Prünster, B., Ziegler, D., Marsalek, A. & Reiter, A. (2016). Balancing utility and security: Securing cloud federations of public entities. In C. Debruyne, H. Panetto, R. Meersman, T. Dillon, e. Kühn, D. O'Sullivan & C. A. Ardagna (Eds.), *On the move to meaningful internet systems: Otm 2016 conferences* (pp. 943–961). Cham: Springer International Publishing.

Tunggal, A. T. (2019, November 20). The 29 biggest data breaches: Updated for 2019. Retrieved November 29, 2019, from https://www.upguard.com/blog/biggest-data-breaches

Turner, R. (2016, April 7). Api security: A disjointed affair: Cios/cisos need to know how api security is managed within their own organizations. Retrieved September 16, 2019, from https://resources.distilnetworks.com/i/660988-ovum-survey-on-api-security

Valvik, R. A. (2012, February 12). Security api for java me: Securexdata. Retrieved September 16, 2019, from http://bora.uib.no/handle/1956/6078

Wijayarathna, C. & Arachchilage, N. A. G. (2019). Why johnny can't develop a secure application? a usability analysis of java secure socket extension api. *Computers & Security*, *80*, 54–73. doi:https://doi.org/10.1016/j.cose.2018.09.007

Zhang, Y., Liu, Q., Luo, Q. & Wang, X. (2015). Xas: Cross-api scripting attacks in social ecosystems. *Science China Information Sciences*, *58*(1), 1–14. doi:10.1007/s11432-014-5145-1

5 Appendix

For verification purposes I have included a list of the papers selection in Appendix 5.1. In Appendix 5.2 a detailed overview of the distribution of sentiment is given, and in Appendix 5.3 the distribution of methodologies. IDs are included for reference.

5.1 Papers Selection

Table 5 gives an overview of the papers selection. See References chapter for details.

Table 5: The table lists the papers in my selection.

ID	Title	Author(s)	Year
1	Security API for Java ME: secureXdata	R. Valvik	2012
2	API Security: whose job is it anyway?	J. Macy	2018a
3	API Security: A Disjointed Affair	R. Turner	2016
4	Adaptive security architecture for protecting RESTful web services in enterprise computing environment	M. Beer, M. Hassan	2018
5	A Multi-Vocal Review of Security Orchestration	C. Islam et al.	2019
6	Precise Android API Protection Mapping Derivation and Reasoning	Y. Aafer et al.	2018
7	A New Malware Detection System Using Machine Learning Techniques for API Call Sequences	M. Jerlin, K. Marimuthu	2018
8	Computability evaluation of RESTful API using Primitive Recursive Function	R. Padmanaban et al.	2018
9	How to build a secure API gateway	J. Macy	2018b
10	API vulnerabilities: Current status and dependencies	T. Bhuiyan et al.	2018
11	Why Johnny can't develop a secure application? A usability analysis of Java Secure Socket	C. Wijayarathna, N. Arachchilage	2019

Continued on next page

ID	Title	Author(s)	Year
	Extension API		
12	Improving API usability	B. Myers, J. Stylos	2016
13	Integrity of intention (a theory of types for security APIs)	M. Bond, J. Clulow	2006
14	On the Security Expressiveness of REST-Based API Definition Languages	H. V. Nguyen et al.	2017
15	Security Analysis of the W3C Web Cryptography API	K. Cairns et al.	2016
16	A Security Policy Oracle: Detecting Security Holes Using Multiple API Implementations	V. Srivastava et al.	2011
17	XAS: Cross-API scripting attacks in social ecosystems	Y. Zhang et al.	2015
18	Most Websites Don't Need to Vibrate: A Cost-Benefit Approach to Improving Browser Security	P. Snyder et al.	2017
19	Balancing Utility and Security: Securing Cloud Federations of Public Entities	B. Suzic et al.	2016
20	A generic security API for symmetric key management on cryptographic devices	V. Cortier, G. Steel	2014
21	A Roadmap for High Assurance Cryptography	H. Halpin	2018
22	Malware classification based on API calls and behaviour analysis	A. Pektaş, T. Acarman	2018

5.2 Distribution of Sentiments and Methodologies

Table 6 shows the distribution of sentiments in the papers selection. Table 7 shows the distribution of methodologies in the papers selection.

Table 6: The table illustrates how the different types of sentiments observed in the papers seem to vary over the years. I have included the IDs for reference purposes.

Sentiment	**Observed Years**	**IDs**
Enthusiastic	2018, 2017, 2016, 2015, 2014, 2012, 2006	1, 4, 6, 7, 13, 15, 17, 19, 20, 22
Concerned	2017, 2018	2, 9, 14
Focused on exposing or reducing threats or infrastructural exposure in general	2018	2, 9
Raises awareness on key findings in a survey	2016	3
Raises awareness on the proper usage and usability issues or bugs resulting in security weaknesses of APIs	2019, 2018, 2016, 2011	10, 11, 12, 16
Raises awareness on balancing user access to core browser features via web API standards	2017	18
Somewhat neutral / professional / seemingly objective	2019, 2018	5, 8
Frustrated, with little hope	2018	21

Table 7: The table shows the applied methods by frequency. I have included the IDs for reference purposes.

Methods	**Frequency**	**IDs**
Observation	1	11
Review	1	5
Survey	1	3
Test	14	1, 6, 7, 8, 10, 13, 14, 15, 16, 17, 18, 19, 20, 22
Using literature	5	2, 4, 9, 12, 21