European Commission - Horizon 2020 DS-07-2017
Cybersecurity PPP: Addressing Advanced Cyber Security Threats and Threat Actors

REactively Defending against Advanced Cybersecurity Threats [†]

WP2: Attack Scenarios
Deliverable D2.1: Threat Models

Abstract: Security issues related to digital information are now prevalent and affect computer systems in several different ways. Securing systems involve, in most of the times, a combination of techniques that target very specific attacks. For instance, protecting a web server from exploitation needs a series of methodologies, which are applied to different layers of the affected software: (a) natively to the server's software, (b) to the web application, and, very likely, (c) to the network that the web server connects to.

Therefore, for any security discussion, it is vital to sort out all of the relevant contexts. Technically, this is equivalent to list all the *threat models* that make sense for a particular defense or attack. In this Deliverable, we collect a list of threat reports from major IT vendors and we focus on the relative threat models that are aligned with the purposes of *ReAct*. We argue that these threat models are very timely and important, since they affect a large fraction of cyber attacks, as observed, during the last couple of years.

Contractual Date of Upload	November 2018
Actual Date of Upload	November 2018
Deliverable Dissemination Level	Public
Editor	Elias Athanasopoulos
Contributors	All *ReAct* partners
Quality Assurance	Herbert Bos

[†] This project is funded by the European Commission (Horizon 2020 - DS-07-2017) under Grant agreement no: 786669.

The *ReAct* Consortium

FORTH	Coordinator	Greece
STICHTING VU	Beneficiary	The Netherlands
UNIVERSITY OF CYPRUS	Beneficiary	Cyprus
EURECOM	Beneficiary	France
RUHR-UNIVERSITAET BOCHUM	Beneficiary	Germany
SYMANTEC	Beneficiary	France

Document Revisions & Quality Assurance

Internal Reviewers

1. Cristiano Giuffrida

Revisions

Ver.	Date	By	Overview
1.0.2	27/11/2018	*Elias Athanasopoulos*	Addressed review comments – final pre-release before submission.
1.0.1	14/11/2018	*Elias Athanasopoulos*	Release to the Review Committee.
1.0.0	2/11/2018	*Elias Athanasopoulos*	First draft.

Contents

1 Introduction **9**
1.1 Threat Models 10

2 Threat Models **11**
2.1 What is a Threat Model? 11
2.2 Example Threat Models 12
2.2.1 Passive and Active Man-in-the-Middle 12
2.2.2 Control-flow and Code-reuse Attacks 14
2.2.3 Beyond Control-flow Attacks 16

3 Threat Reports **17**
3.1 Symantec 2018 Internet Security Threat Report 17
3.1.1 Coin Mining 17
3.1.2 Ransomware 18
3.1.3 Targeted attack groups 18
3.1.4 “Living-off-the-land” techniques 19
3.1.5 Internet of Things 19
3.1.6 Predictions 20
3.2 Cisco 2018 Annual Cybersecurity Report 20
3.2.1 Internet of Things 20
3.2.2 Leak paths 21
3.2.3 Encryption and Legitimate Cloud Resources used for Malicious Activity 21
3.2.4 Predictions 22
3.3 Akamai 2018 State of the Internet Security Report 22
3.3.1 Financial Trojans 22
3.3.2 JS Miner 22
3.3.3 Mirai malware 22

3.3.4 Mobile divices and Loapi . . . 23
3.3.5 Predictions . . . 23
3.4 Europol 2018 Internet Organised Crime Threat Assessment Report . . . 23
3.4.1 Ransomware . . . 23
3.4.2 Cryptocurrency Miners . . . 23
3.4.3 DDoS . . . 24
3.4.4 Predictions . . . 24
3.5 Memory Errors today . . . 24
3.5.1 Trends in Memory Errors . . . 26
3.6 Summary . . . 32

4 ReAct Threat Models **35**
4.1 Threat Models . . . 35
4.2 Short Technical Overview . . . 39
4.3 Specific Threat Models and Assumptions . . . 40
4.3.1 Scanning and Probing . . . 40
4.3.2 Forecasting . . . 41
4.3.3 Patching . . . 42
4.3.4 Isolation . . . 42
4.3.5 Forensic Analysis . . . 43
4.4 Summary of Attack Scenarios . . . 43

5 Conclusion **49**

List of Figures

3.1 Memory-error vulnerabilities from 1997 to 2017. It is important to notice that, despite the many defenses deployed and software evolution, memory errors are still prevalent. 26

3.2 Memory-error vulnerabilities and exploits throughout the years. It is noticeable that the trend of exploits follow the trend of memory-error vulnerabilities with a small delay. A very clear observation that *patching* of modern software plays an important role for security. 27

3.3 Reported memory-error vulnerabilities compared to *all* vulnerabilities reported. The memory-error bugs pose only a small fraction of all reported vulnerabilities, however, this fraction: (a) seems stable, and (b) does not seem to go away anytime soon. 28

3.4 Reported memory-error vulnerabilities compared to *all* vulnerabilities, as well to the ones related to web. It is likely that several web-related vulnerabilities contribute to an excessive amount of vulnerabilities reported. 29

3.5 Relative number of memory-error vulnerabilities and exploits, subject to the total number of reported vulnerabilities and available exploits. 30

3.6 This figure shows the relative number of memory-error and web vulnerabilities and exploits, subject to their totals. . . . 30

3.7 The number of reported memory-error vulnerabilities, categorized in stack-based, heap-based, integer, pointer, format string, and other issues. 31

3.8 The number of reported memory-error exploits, categorized in stack-based, heap-based, integer, pointer, format string, and other issues. 31

4.1 *ReAct* generic threat model. 36
4.2 *ReAct* advanced threat model. 38

1 Introduction

Information security is today a very complex domain of research, which combines knowledge from a broad range of entirely different topics. From low-level operating systems and micro-architectures to user behavior and psychology. A security researcher can evolve information security by taking different paths that achieve different goals. Building a new methodology or technique for offering holistic security is perhaps not a realistic option. To the contrary, security research attempts to solve very specific problems, which connect to very specific goals. In several cases, security research concludes to new methodologies that do not entirely solve a problem, but, rather, they *raise the bar*. That is, a technique can be practically valuable even if it makes things *just harder* for the attacker.

Additionally, security research, as it happens with all science, attempts to approach problems using the appropriate rationale. For designing and implementing defenses, we need to narrow down the type of attacks that these defenses are suitable for. Recall, that nowadays attacks are complicated and, in most of the cases, they combine different tricks, they abuse different layers of operation, and most likely they take into account several different defenses in place.

Narrowing down the relevant attacks for a defense is most commonly abbreviated as the *threat model*. The definition of the threat model is very important for understanding the whole setting and, more precisely, the impact of the defense. Without a proper definition of a threat model, any discussion of a possible defense is hard. We cannot reason the effectiveness and the limitations of the in question methodology.

Defining the threat models is not always trivial. Here is representative list of challenges. First, the threat model should be complete. That is, the definition of the threat model should be such that there is no ambiguity or vagueness of the attacks included. Second, if several techniques are composed, such as the case of *ReAct*, a set of threat models should be appropri-

ately selected efficiently, without unnecessary duplication and overlapping. Finally, the resulted threat models should be timely aligned. Dealing with threat models of the past is not productive.

In this Deliverable, we list all threat models that are relevant to *ReAct*. This is done in the following four phases:

- We first discuss what *is* a threat model, and go through some very common threat models for making the reader more familiar (Section 2).
- We then present a list of threat reports, created by major IT vendors in Information Security for identifying important real-world attacks that have been recorded recently (Section 3).
- We then produce *generic* threat models that encapsulate a large fraction of real security incidents (Section 4).
- We finally list very precise threat models and assumptions for all individual techniques produced in the three technical work-packages of *ReAct*, namely WP3, WP4, and WP5 (Section 4).

Some *key* observations that stem from this deliverable are the following.

- Attackers may be interested in new malicious activities, which are aligned with the current technological evolution. As an example attackers may compromise servers for crypto-mining power (i.e., use the compromised hosts for mining digital coins, such as Bitcoin).
- Nevertheless, no matter the end goal, attackers *still* use memory errors to compromise machines. This happens, transparently, without social-engineering tricks.
- Despite the many defenses against software exploitation that is based on exploiting memory safety, we observe that (a) memory errors are still prevalent, (b) memory errors can be still exploited successfully. One *key* attack incident, as reported by all collected threat reports, is the success of ransomware (such as WannaCry and Petya), which was delivered by exploiting memory errors.

1.1 Threat Models

The deliverable is named *Threat Models* for conveying that the *attack setting* is discussed. The threat model, as explained later, is the mechanism that defines a setting of possible attacks. Therefore, we use threat models for defining the attack setting and, once this is defined, then several attack scenarios that make sense can be created.

2 Threat Models

In this part, we attempt to make the reader more familiar with a threat model. We first begin with a sort, abstract, discussion of what is included in a threat model, and we later discuss a few very common and representative threat models. The examples presented here are not necessarily threat models relative to the goals of *ReAct*.

2.1 What is a Threat Model?

A threat model is used broadly in security research to define a meaningful context. In other words, the threat model creates the necessary setting for discussing possible attacks or defenses. Although there is no formal definition of a threat model, and several threat models are very differently stated, the basic ingredients are the following. A threat model should include:

- A list with the attacker's capabilities,
- A list with the attacker's goal(s),
- Often, a list with the defenses that are in place,
- Often, a list with the affected risks of the target system using security requirements (CIA).

When security requirements are used, we indicate that these are the standard ones as defined in the CIA model, namely Confidentiality, Integrity and Availability, as well as the commonly used ones (Authentication, Authorization, Non-repudiation, etc.).

It is crucial to understand that the threat model may be generic or very detailed, depending on the expressiveness we want to stress. For instance, the attacker's capabilities can be expressed using abstract primitives, i.e., *the*

attacker can inspect network traffic, or very detailed ones, i.e., *the attacker can inspect network traffic of the protocol X, which is exchanged by endpoint A and endpoint B.*

Both types of threat models are useful. Generic threat models are used to exclude totally non-related attacks from the setting. For instance, a system-based threat model which explores micro-architectural attacks can exclude the user factor which dominates completely in different attacks, such as phishing [14]. On the other hand, very specific threat models are useful for understanding the technological depth of a particular attack or defense. For instance, a threat model may involve a very precise type of architecture – the Intel family of processors that support particular virtualization functions, such as Extended Page Tables– in order to stress how this technology plays a crucial role for implementing a defense.

In Chapter 4 we discuss both generic threat models that cover the work of *ReAct*, as well as very specific threat models and assumptions that define the context of the tools created as part of WP3, WP4 and WP5.

Last but not least, threat models help us to understand that *"Security isn't a scalar. It doesn't make sense to ask 'Is device X secure?' without a context: 'secure against whom and in what environment?'"*[5]

2.2 Example Threat Models

Here we discuss some examples of threat models. These examples are not necessary related to *ReAct*; their use here is just indicative for conveying the role of the threat model.

2.2.1 Passive and Active Man-in-the-Middle

In network security, we are usually interested in securing network connections. For most of the cases, these connections exchange data in an encrypted form, using traditional cryptographic algorithms. Of course, several different properties play a role here. A network connection is fairly vague, since we have not defined the network layer we are interested in or the protocols in play.

Thus, we can define a generic threat model for a *passive Man-in-the-Middle (MitM) attacker*, as follows:

1. An attacker that can passively monitor network packets exchanged between two parties,

2. Attacker wants to reveal the conversation,

3. Conversation is encrypted using the cryptosystem X,

4. Confidentiality can be affected if the attacker can break cryptosystem X,

5. Integrity, and Availability cannot be affected.

Following the discussion in the beginning of this chapter, the above threat model maps to the following entities. In (1) we list the attacker capabilities. In our case, the attacker is constrained. The attacker can only monitor the network, but cannot modify, inject or drop network packets. In (2) we list the attacker goal, which is to reveal the contents of the conversation. Notice, that there are *different* threat models, which target disclosure of anonymity. In detail, the attack could be interested just to reveal which endpoints are part of the network communication and not the contents of the communication per se. In (3) we list potential defenses in place, namely that the conversation is not in plain, but cryptography is used for hiding the data exchange. Finally, in (4) and (5) we list which security requirements are likely to be affected and which not, respectively.

One step further, the just mentioned threat model could be extended to support a stronger attacker and, in particular, one that beyond passively monitoring the network can modify the data exchanged. This *active attacker* can inject, modify existing or drop network packets from the communication link. Here is the threat model for an *active Man-in-the-Middle (MitM) attacker*:

1. An attacker that can actively monitor, inject, modify, or drop network packets exchanged between two parties,

2. Attacker wants to reveal the conversation,

3. Conversation is encrypted using the cryptosystem X,

4. Confidentiality can be affected if attacker can break cryptosystem X,

5. Integrity, and Availability can be affected.

At this point, we would like to stress two points. First, we can see that the definitions of the two threat models are very close. However, in practice, the two threat models describe an entirely different class of an attacker. The active network attacker is much stronger than the passive one. Second, the transition from a passive to active attacker has consequences. In the second threat model all basic security requirements are affected. Again, several specifics are left out, on purpose, from the threat model definition. For instance, the cryptosystem used is not revealed; several cryptosystems incorporate integrity checks.

To make the discussion more clear, the following threat model is aligned with the active network attacker threat model and describes an attack against RC4, a very well-known stream cipher, when used in TLS [4]:

1. An attacker that can actively monitor, inject, modify, or drop network packets exchanged between two parties,
2. An attacker can exploit statistical biases in RC4 and launch a ciphertext only plain recovery attack,
3. Attacker wants to reveal the conversation,
4. Conversation is encrypted using TLS and specifically RC4 as an encryption cipher,
5. Confidentiality can be affected.

Now, this threat model, lists very specific technologies in place, and the attacker is concentrated in launching a very specific attack by targeting an encryption algorithm, which is offered as an option by TLS.

It is crucial to stress that both types of threat models are useful in research. In *ReAct* we use threat models of different expressiveness, as well. We use generic threat models that describe classes of attacks, and then we present very precise threat models for the techniques developed.

2.2.2 Control-flow and Code-reuse Attacks

We continue with a discussion of threat models compatible and aligned with system and software security, rather than network security. The following threat model discusses an attacker that can launch control-flow attacks.

1. An attacker that can interact with a target vulnerable program by sending malicious inputs locally or over the network,
2. The attacker can leverage vulnerabilities to acquire an arbitrary write and read primitive,
3. The attacker wants to redirect the control flow of the program by overwriting control data using the write primitive,
4. The stack/heap is not executable and ASLR is in place,
5. The target program can be exploited and controlled by the attacker by utilizing the write/read primitives for launching a ROP attack.

The aforementioned threat model is fairly compact, and it contains technical information, which is not really evident during a first read. We now elaborate more on all items included in the particular threat model, since this one is very close to threat models relevant with *ReAct*.

In (1), (2), and (3) we list the attacker capabilities and goal. We explicitly assume that the attacker is interacting with a vulnerable program. This

is very aligned with the spirit of *ReAct*, since we expect that, no matter the techniques involved for producing software, if programs are developed in an unsafe system (C/C++) then it is likely that they contain vulnerabilities. Furthermore, the attacker can interact with the vulnerable program by sending (malicious) inputs. These inputs can be sent by a local attacker that aims at acquiring higher privileges in the system, or by a remote attacker that aims at compromising the host. The attacker acquires *primitives* by triggering vulnerabilities through (malicious) inputs. This is also important, since the threat model assumes a fairly strong attacker, who, due to bugs, has the ability to *write* and *read* all process' memory. The attacker is now able to program the target by just sending inputs, since each input can exercise the acquired read/write primitives. The vulnerable program can execute malicious commands (for instance, download malware or create a backdoor in the system) or exfiltrate sensitive information (e.g., a password).

The threat model, through items (1)-(3), defines a strong attacker, however, in (4) and (5) the attacker is further constrained in terms of available techniques. According to (4) the stack/heap is not executable. This assumption constrains the attacker and prevents them to launch a simple code-injection attack [3]. Instead, the attacker needs to launch a code-reuse attack, possibly through Return-oriented Programming (ROP) [27]. Furthermore, in (5) another assumption makes the attack even more complicated. The fact that ASLR [24] is in place mandates that the attacker needs several vulnerabilities for attacking the program and, at least one of them, should be a vulnerability that explicitly gives the attacker a, possibly interactive [29], read primitive.

Therefore, we can expand the threat model to include all necessary information by stressing the following properties.

1. The attacker targets programs written in unsafe systems, such as C and C++,
2. These programs contain memory-based vulnerabilities, which, if triggered, give the attacker read/write primitives,
3. The attacker can leverage the read and write primitives to control the target program,
4. The attacker must use certain techniques to overcome a set of defenses,
5. The defenses in place (NX-bit, ASLR, stack canaries) coerce the attacker to use their primitives for revealing the code layout of the target and perform code reuse for performing any useful attack.

The aforementioned description is not a threat model, but encapsulates, in high-level, all steps an attacker should take in order to leverage a threat model that describes control-flow attacks.

2.2.3 Beyond Control-flow Attacks

Not all system/software attacks follow the threat model presented in Section 2.2.2. For instance, web applications suffer from their own vulnerabilities and web attacks are delivered using exploits of different mechanics. Another example is side-channel attacks, which can compromise a system without leveraging *any* software vulnerability. Instead, side-channel attacks leverage certain micro-architectural effects that can be observed macroscopically in a system, and can be leveraged to infer useful information, that may contain secrets.

Both web attacks and side-channel attacks are just examples of attack scenarios that are out of the main scope of *ReAct*[1]. In principle, there are several threat models that include attacks that cannot be captured with the techniques developed within *ReAct*. Nevertheless, as we explore in Chapter 3, several real-world incidents, as demonstrated by established IT vendors, can be encapsulated in a set of threat models that are very close to the one discussed in Section 2.2.2.

[1] Although, T2.2 (Next Generation Attacks) explores some advanced attacks that are side-channel based.

3 Threat Reports

The cyber-crime threat landscape is well known to be a fast changing environment manifested by a diverse pool of attacks. In order to fully grasp the current trends of attacking the real world, we studied several threat reports from various worldwide established organizations. Specifically, we assembled a list of reports composed by the following:

- Symantec 2018 Internet Security Threat Report,
- Cisco 2018 Annual Cybersecurity Report,
- Akamai 2018 State of the Internet Security Report,
- Europol 2018 Internet Organised Crime Threat Assessment Report.

In the following part we discuss the major security incidents of 2018, as recorded in the aforementioned reports. For each report, we highlight specific findings and observations that are important for *ReAct*, as well as predictions for the near future. Additionally, in the last part of this chapter, we expand on recent trends of memory errors; a class of program errors that allow sever attacks, as documented by the studied reports.

3.1 Symantec 2018 Internet Security Threat Report

3.1.1 Coin Mining

Coin Mining [20] is a term that is used very often in the world of cryptocurrency. Simply put, by saying coin mining we mean the act of getting rewarded with cryptocurrency for solving a cryptographic puzzle, needed for verifying other transactions done on the blockchain [23]. The problem here is that this procedure needs a tremendous volume of computational processing power. For this reason attackers install these little programs called

miners on their victims' computers in order to take advantage of their hardware capabilities and use them for coin mining. This kind of attack attracted many cybercriminals as appears of 8,500% increase in coin miner detections on endpoint machines during the last two years.

This sudden increase happened because of the tremendous rise value of cryptocurencies. These programs are either file-based which infect users machines or browser-based meaning that are executed when the user visits an infected website. In both cases the victims hardware is been overused in the background without their knowledge. Coin mining is considered illegal only when it's done without the user's knowledge and consent.

We stress here that coin mining is the *goal* of the attacker, which can be satisfied when the user, using social engineering methods, or when the system, using software exploitation, is compromised.

3.1.2 Ransomware

Another attack that saw a huge increase during the last year is the one of Ransomware [18]. Ransomware attack as the name suggests, is the attack where victims are blackmailed into paying money (in cryptocurrency, often Bitcoin). Basically the attackers by exploiting operating system vulnerabilities, encrypt all their victims' data and demand for the said ransom in exchange for the decryption key. These kind of attacks have been around since 2015 but attracted the security researches attention with the WannaCry and Petya/NotPetya attacks because of their considerable impact over the last year.

An interesting issue that the report points out is the fact that cyber criminals utilize the Ransomware attacks in order to hide other attacks. One example of this strategy is the Petya/NotPetya attack. As it was later reported, there was no way to decrypt the encrypted files because of the way the malware was originally engineered. For this reason its been assumed that that Petya/NotPetya was actually a disk wiper disguised as a Ramsonware attack.

3.1.3 Targeted attack groups

Target attack groups are organized groups of security experts that their main goal is to execute targeted attacks against organizations. These organized groups are state sponsored which as the report shows their main motive is intelligence gathering rather disruption or sabotage. In the following section we discuss two intrusion techniques that are often used by targeted attack groups in order to penetrate an organization.

3.1.3.1 Spear Phishing

Spear Phishing is a sub-category of phishing [14]. Instead of sending out a generic email in order to extract information from random recipients, Spear phishing attackers send targeted emails usually representing a high raking or known individual to their target. These emails are carefully written in order to look like a legitimate email. As the report mentions, spear phishing is used by over 71% of the targeted attack groups which makes it the most widely used infection vector.

3.1.3.2 Watering holes

Another serious attack used by over 24% of the targeted attack groups is watering holes which also targets specific individuals. Attackers compromise websites which are related to their target's interest so its more likely for the target to visit that website. Once their target visits the compromised website, the target's machine gets infected. These websites are compromised without their owners knowledge and despite the fact that this method can infect more users other than the target, the attackers tend to check if the visitor's IP matches the target's IP in order to reduce the chances of collateral damage.

3.1.4 "Living-off-the-land" techniques

"Living-off-the-land" techniques are various modern methods that are replacing common tactics like exploiting zero day vulnerabilities. For instance, one living-off-the-land technique consists of utilizing legitimate network administration and operation system features instead of installing exploit kits. In addition, another example that belongs in this category is the attacks against supply chains.

Supply-chains attacks consist of infecting a third-party software in order to infiltrate an organization using that software where would be otherwise impossible to achieve a direct attack. One example of a supply-chain attack is the one of the Petya/NotPetya Ransomware attack. By infecting the supply chain, namely an accounting software, attackers achieved to infiltrate various organizations in Ukraine.

One reason for this shift in attacks is due to the fact that by utilizing already installed software, less traces are left behind and as a result it becomes difficult to trace the source of the attack. In addition, as industrial software becomes more secure, so does finding a zero-day vulnerability.

3.1.5 Internet of Things

As the adoption of Internet of Things(IoT) devices increases throughout the years so does the number of attacks against these devices. As the report

mentions, from 2016 to 2017 the attacks against IoT devices have been increased more than 600%. One major example of how IoT alters the cyber security threat landscape is the Mirai botnet [6] which has disrupted major organizations with the use of DDoS attacks. Moreover what makes IoT devices even more threatening is the fact that they have poor security countermeasures, like unpatched vulnerabilities.

3.1.6 Predictions

The report's predictions firstly focus on the modern processor chip vulnerabilities known as Meltdown [22] and Spectre [19] as these two had a major impact at the start of 2018. Moreover the report points out that is likely we see an increase of self-propagating threats like WannaCry and Petya/NotPetya as well as the techniques that helped the spreading of these two threats, namely the supply-chain attack which falls under the "Living-off-the-land" technique. This is particularly interesting for *ReAct*, since the reactive defenses delivered in this project target exactly this: self-propagation through software exploitation.

The report points out that these techniques are increasing in use because are more appealing to the cybercriminals as it helps them work under the radar with less chances of getting caught. In addition, despite the fact that IoT attacks weren't in the spotlight in 2017 that doesn't mean that this threat was over. On the contrary as Internet of Things attackers fight over the same pool of targeted devices and as a result they are searching for other IoT devices other than routers and modems that they have been already attacked. Finally, the last prediction of this report is that the coin-miner activity will increase throughout 2018 and especially coin miners targeting organizations in order to harness their massive servers' power.

3.2 Cisco 2018 Annual Cybersecurity Report

3.2.1 Internet of Things

As it is well known Internet of Things (IoT) devices are massively adopted from the general population and especially from organizations. Organizations tend to install these devices on their network without considering the many security threats and vulnerabilities that these devices might cause. Unfortunately, as the report states, most organizations leave IoT devices unmonitored and unpatched which results in massive network vulnerabilities and easy targets for attackers. We stress here that *patching*, through *selective fortification*, is one of the core components of *ReAct*.

The report also mentions that a Cisco partner Qualys, tested a sample of 7,328 IoT devices for several known threats and found that 83% are

vulnerable to these threats. Including the Devil's Ivy, a vulnerability in the gSOAP library, used in the Mirai malware [6].

3.2.1.1 DDoS

As stated earlier IoT devices which are easier to get exploited than PCs and are getting increasingly deployed by organizations. Consequently, attackers take control of these devices and deploy large scale DDoS. An example of a IoT botnet is the one of Mirai which is well known for DDoS attacks against high-profile organizations like Github, Twitter, Reddit, Netflix, Airbnb and many others [6].

3.2.2 Leak paths

Leak paths - definition given by a Cisco partner Lumeta, is an unauthorized or misconfigured connection created to the internet on an enterprise network. As the report points out, Lumeta estimates 40% of enterprise dynamic networks lack real time awareness for security teams.

3.2.3 Encryption and Legitimate Cloud Resources used for Malicious Activity

Attackers abuse cloud resources in order to achieve Command and Control (C2). C2 server is a computer controlled by a malicious actor which is used for receiving data from a network by sending commands to the compromised computer. Cybercriminals register new accounts in cloud providers services and take advantage of their features like encryption used for C2 protocols, setting up a publicly accessible web page and finally using the IP addresses provided by these services which ensure their anonymity; these are some of the benefits of abusing legitimate cloud resources.

Especially important feature of the cloud resources is the encryption of the traffic. As the report states, Cybercriminals are concealing their malicious traffic with the use of encryption and this is expected to rise in 2018. This technique makes even more difficult for organizations to defend their infrastructure from attackers as it is almost impossible to distinguish genuine from malicious traffic coming from legitimate cloud resources. In order for organizations to detect encrypted malicious activity coming from legitimate cloud resources as the report points, they started using machine learning techniques which give a better chance of successfully detecting this kind of traffic.

3.2.4 Predictions

This report's predictions mainly focus on how the cybercriminals will keep adapting in order to evade detection. Namely by using legitimate internet services and encryption, they are able to work under the radar without getting caught. In addition it is likely that the malware produced by cybercriminals will get even more difficult to notice, even for known threats.

3.3 Akamai 2018 State of the Internet Security Report

3.3.1 Financial Trojans

As the report states, there has been a shift of the financial Trojans from targeting peoples' financial credentials to social interactions' data. What does this mean is that, the value of peoples' interactions data are more valuable than their bank's credentials to malicious actors. The use of people's data can vary from promoting sponsored news to cyber-espionage.

3.3.2 JS Miner

As mentioned in previous reports so does this report acknowledges the massive impact that the rise of cryptocurencies' value had to the attraction of cybercriminals for creating numerous coin-mining malware [20]. This report specifically focuses to the web-based coin-miners written in JavaScript. Basically when a user visits a coin-mining hosted website, its device's computational processing powers get taken advantage for coin-mining [7].

The report states that some websites have coin-mining ready code and ask visitors for their consent for using their devices for coin-mining. This technique is used as a revenue model substitute to advertising but other websites do this without notify the visitor and even some times even the owner of the website isn't aware that the website god infected with coin-mining malware.

3.3.3 Mirai malware

Mirai malware [6] attacks had a tremendous impact because of its catastrophic capabilities. Throughout 2017 major DDoS attacks against high-profile organizations have been recorded. Nevertheless, the report states that IoT devices infected with the Mirai malware have evolved from just initiating DDoS attack to Ramsonware malware distributors

3.3.4 Mobile divices and Loapi

Cybercriminals have leveraged the opportunity of the widely adoption of mobile devices and created a trojan, called Loapi which is capable of a great ranges of malicious activities [34]. The infected device can participate in DDoS attacks, mine cryptocurrencies, send malicious SMS and display unwanted advertisements. This malware is a great example of how mobile devices' malware evolves as past created malware could only perform only one kind of malicious activity in contrast with Loapi's numerous capabilities.

3.3.5 Predictions

This reports' predictions firstly stresses out the appearance of Loapi trojan which is capable of a great range of activities and as the mobile devices OS are growing so does the potentials for malicious activity. In addition it is likely to see new Trojans like Loapi capable of performing numerous malicious activities like DDoS and coin mining.

Moreover, the report points out the exploitation of social networks in which cybercriminals hack social networks accounts for phishing, cyber-espionage and finally fake-news spread. Last but not least, like Symantec's report so does this report predict the increase of coin miners and especially the web-based ones which are predicted to constitute the main revenue resource for websites but with the threat of leaving users' devices vulnerable to coin miners originated from malicious sources.

3.4 Europol 2018 Internet Organised Crime Threat Assessment Report

3.4.1 Ransomware

As the other reports state so does this particular one highlights that ransomware [18] had a massive impact on the cybersecurity landscape. Especially as a result of its self-replication capabilities. As the report mentions, the WannaCry and NotPetya attacks affected an estimated 300,000 victims worldwide with *4 USD billion in losses*. Now, with the ease of using ransomware-as-a-service its more widely accessible than ever for anyone to launch a ramsonware attack even with limited programming skills.

3.4.2 Cryptocurrency Miners

Cryptocurrency miners are also discussed as *cryptojacking* in this report [20]. They have attracted many cybercriminals as it constitutes a form of revenue model. Despite the fact that cryptocurrency mining is not illegal, itself, cybercriminals hack legitimate websites to cryptojack their visitors. In addition

cryptomining malware constitutes another way of cybercriminals to cryptojack their victims. When this malware is downloaded to the victim's device it starts exploiting its hardware resources for cryptocurrency mining. Being an emerging threat, cryptocurrency is still new to the cybersecurity threat landscape with insufficient law enforcement as it is still in a grey area.

3.4.3 DDoS

Distributed Denial-of-Service (DDoS) is an attack in which the attacker tries to flood its target's network with request in order to make it unavailable. This attack usually is carried out using a botnet, namely an army of PCs [25] or in as it transformed in modern times, an army of Internet of Things (IoT) devices [6]. As the report states, DDoS attacks are one of the most commonly reported cyber-attacks, second only to malware attacks. Furthermore, DDoS attacks had a huge increase and are accounted for over 70% of all incidents compromising network integrity. The rise in DDoS attacks is due to the ease of anyone, even an unskilled individual, of launching a DDoS attack as many tools providing such services have been openly advertised and easily used.

3.4.4 Predictions

As all of the aforementioned reports pointed out, so does this one predicts that future threats will include ransomware and cryptomining. In particular, it is likely that ransomware attacks will continue to grow as the tools needed are getting more accessible to cybercriminals.

In addition, cryptomining seems like an appealing replacement of ransomware attacks as a revenue model as it is able to function under the radar without explicitly engaging the victim. Furthermore, mobile malware targeted for financial profit is likely to see an increase as more users execute banking activities through their mobile device. Finally the report states that it is unlikely to see in the future an attack like WannaCry or NotPetya for financial purpose as their massive global attention and the degree of law enforcement does not make up for the profits. This is a prediction, which of course does not rule out similar attacks to WannaCyr/NotPetya by means of exploitation mechanisms.

3.5 Memory Errors today

We have already discussed several threat reports from industrial vendors. From the reports, it is evident that several severe attacks are based on memory errors. Therefore, in this section, we discuss the prevalence of memory

errors today in popular applications. Memory errors is a class of vulnerabilities that may allow attackers to exploit native software [1]. Of course, memory errors is not the only way to exploit software. For instance, web applications can be exploited by using other, non memory related, vulnerabilities.

In the previous part of this chapter we discussed the landscape of real-world incidents as recorded by popular IT vendors. Among the reported incidents, there are *still* several cases where the attack happened due to a memory error (e.g., WannaCry). In this part, we review vulnerabilities as classified by popular repositories in the public domain, namely the National Vulnerability Database (NVD) [2] and exploit-db [3].

Our review attempts to explore the fraction of memory-error vulnerabilities compared to all vulnerabilities reported. This is important, since the core threat model of *ReAct* includes exploitation by leveraging memory errors.

The discussed vulnerabilities have been extracted by the aforementioned sources (NVD and exploit-db) and classified according to their metadata. Here is an overview of the classification. For a more detailed discussion about the methodology, please refer to the memory-errors paper [33].

- **Web**. All vulnerabilities and exploits with a description that includes the following keywords: php, sql, or xss.
- **Stack**. All vulnerabilities and exploits with a description that includes the following keywords: stack-based and stack overflow.
- **Heap**. All vulnerabilities and exploits with a description that includes the following keywords: heap-based, heap overflow, use-after-free, and double free.
- **Integer errors**. All vulnerabilities and exploits with a description that includes the following keywords: integer, signedness, or off-by-one.
- **Dereferenced pointers**. All vulnerabilities and exploits with a description that includes the following keywords: dereference, and dangling pointer.
- **String-formatting errors**. All vulnerabilities and exploits with a description that includes the following keywords: format string.
- **Other**. All vulnerabilities and exploits with a description that includes the following keywords: overflow.

[1] We stress here our phrasing; we have used *may*, since not all vulnerabilities can be used in practice for exploiting software.

[2] https://nvd.nist.gov/vuln/data-feeds#CVE_FEED

[3] https://github.com/offensive-security/exploitdb

The tracking of memory errors is an ongoing open-source project, run by a member of the *ReAct* consortium, and it can be found at: https://www.vvdveen.com/memory-errors/ More details about the scientific approach of tracking memory errors in the wild can be found in the related paper [33].

3.5.1 Trends in Memory Errors

Based on the vulnerability collection and classification we have done as discussed above, we can explore several interesting statistics of memory errors. Here is a short discussion of some of the most important observations.

Memory-error vulnerabilities

In Figure 3.1 we depict the collected vulnerabilities which are classified as memory errors. The reports are from 1997 to 2017. It is important to stress here that despite the attention and research invested in countering software exploitation during the last two decades, memory-error vulnerabilities did not stop. To the contrary, we can see a rise through the years, signifying an effort on finding *new* bugs and.

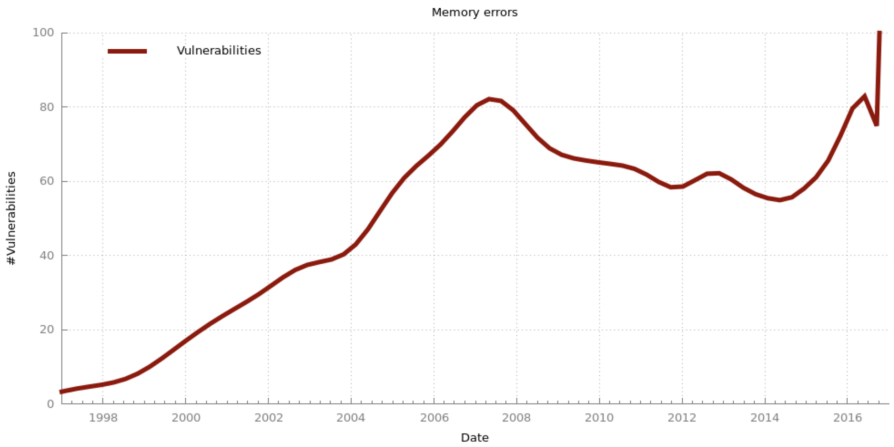

Figure 3.1: Memory-error vulnerabilities from 1997 to 2017. It is important to notice that, despite the many defenses deployed and software evolution, memory errors are still prevalent.

Memory-error exploits

In Figure 3.2 we depict memory-error vulnerabilities and exploits throughout the years. It is noticeable that the trend of exploits follow the trend of memory-error vulnerabilities with a small delay. A very clear observation that *patching* of modern software plays an important role for security.

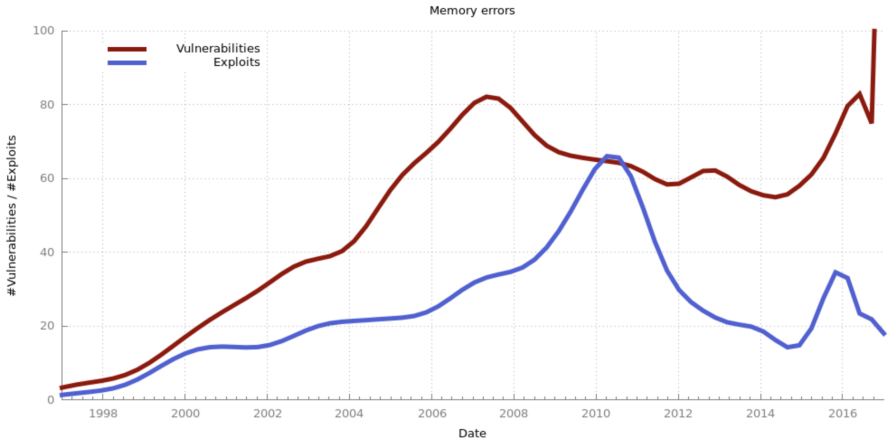

Figure 3.2: Memory-error vulnerabilities and exploits throughout the years. It is noticeable that the trend of exploits follow the trend of memory-error vulnerabilities with a small delay. A very clear observation that *patching* of modern software plays an important role for security.

Memory-error vulnerabilities compared to total

In Figure 3.3 we depict all vulnerabilities reported in parallel with the ones related to memory errors. Clearly, memory errors pose only a fraction of all reported vulnerabilities. However, this is still alarming. Memory errors have a stable presence (compared to *all* bugs) and this trend seems to to be long (over two decades, so far).

Additionally, we can see a large increase of the vulnerabilities reported from 2004 and later. We explore this increase in the next graph.

Memory-error vulnerabilities compared to total/web

In Figure 3.4 we depict the reported memory-error vulnerabilities compared to all vulnerabilities, as well as to the ones related to web. We can see that from 2004 and later there is a large increase in all reported vulnerabilities and, in parallel, there is also a large increase in web-related reported vulnerabilities. It is evident that the relatively low fraction of memory errors compared to all vulnerabilities is not related to reduction of memory errors, but to the increase of the overall reported vulnerabilities. This increase is

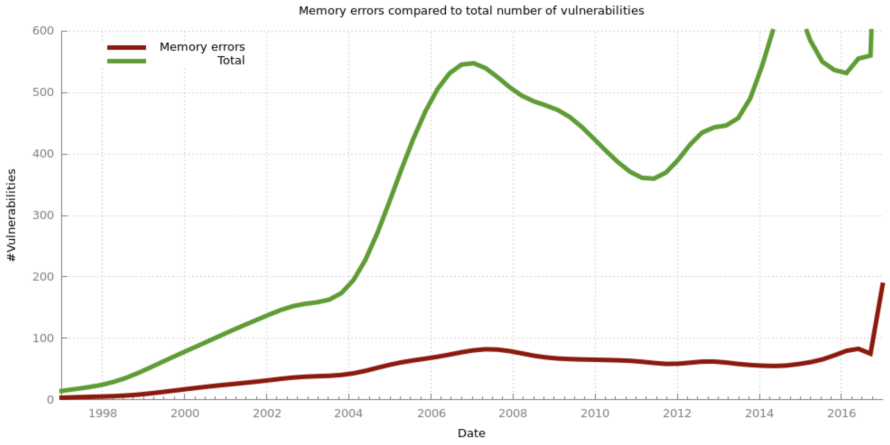

Figure 3.3: Reported memory-error vulnerabilities compared to *all* vulnerabilities reported. The memory-error bugs pose only a small fraction of all reported vulnerabilities, however, this fraction: (a) seems stable, and (b) does not seem to go away anytime soon.

most likely related with the massive popularity of web software during this period of time. From the figure, we can see that there is a clear trend shared by both all vulnerabilities reported and web-based vulnerabilities reported.

Relative number of memory-error vulnerabilities and exploits

In Figure 3.5 we depict the relative number of memory-error vulnerabilities and exploits, subject to the total number of reported vulnerabilities and available exploits. From the figure we can observe that since 2006, memory-errors are responsible for 10 to 20% of all issues, and that exploiting is a bit harder than finding an issue. Although 2013 may have been the start of a downward trend, recent numbers from 2015 and 2016 indicate that memory errors are again fairly popular, despite the many defenses for software that are currently available and the many techniques enabled for fining bugs during development. This is an alarming finding and justifies that the core threat model of *ReAct* aims at defending a severe class of vulnerabilities.

Relative number of memory-error/web vulnerabilities and exploits

In Figure 3.6 we depict the relative number of memory-error and web vulnerabilities and exploits, subject to their totals. We observer that the downward trend in the percentage of memory errors around the year 2003 can solely be contributed to the rise of the world-wide web. This is observation was also discussed in Figure 3.4. Additionally, we see that over the

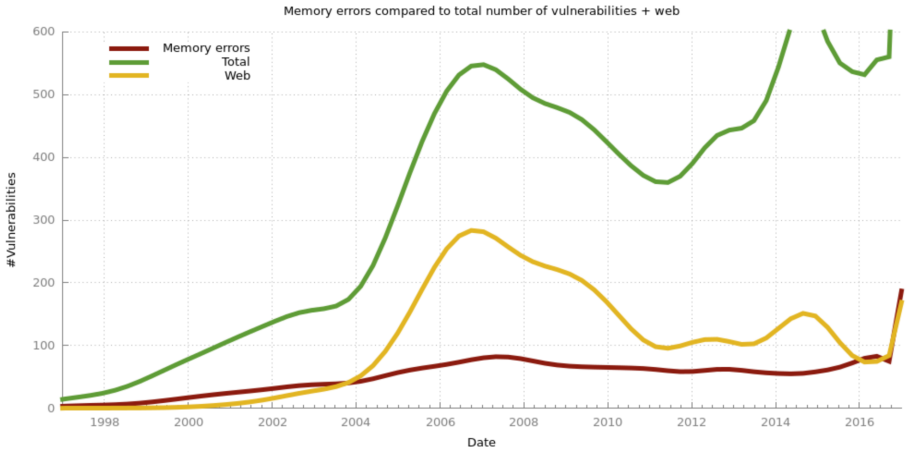

Figure 3.4: Reported memory-error vulnerabilities compared to *all* vulnerabilities, as well to the ones related to web. It is likely that several web-related vulnerabilities contribute to an excessive amount of vulnerabilities reported.

last years, web issues seem to become less dominant, resulting in a slight increase of memory errors.

Types of memory-error vulnerabilities

In Figure 3.7 we depict the number of reported memory-error vulnerabilities, categorized in stack-based, heap-based, integer, pointer, format string, and other issues. As expected, it shows that the stack was a popular attack vector in the early years [3, 12], while recently the heap has become a more prevalent topic [30, 32, 17]. This might indicate that stack-based issues are easier to find and solve automatically, e.g., by compiler extensions [1, 2], binary analysis [9] or the use of shadow stacks [13]. On the other hand, the heap is still much more difficult to reason about and several attacks that are based on hijacking control-data on the heap can be very hard to detect [26, 21].

Types of memory-error exploits

In Figure 3.8 we depict the number of reported memory-error exploits, categorized in stack-based, heap-based, integer, pointer, format string, and other issues. We observer that exploits are slowly catching up with trends in reported vulnerabilities: up until 2015, the stack was still the most popular attack vector, while only recently attackers started to look with more detail into heap attacks. It also shows that format string exploits [11, 28] are basically non-existent.

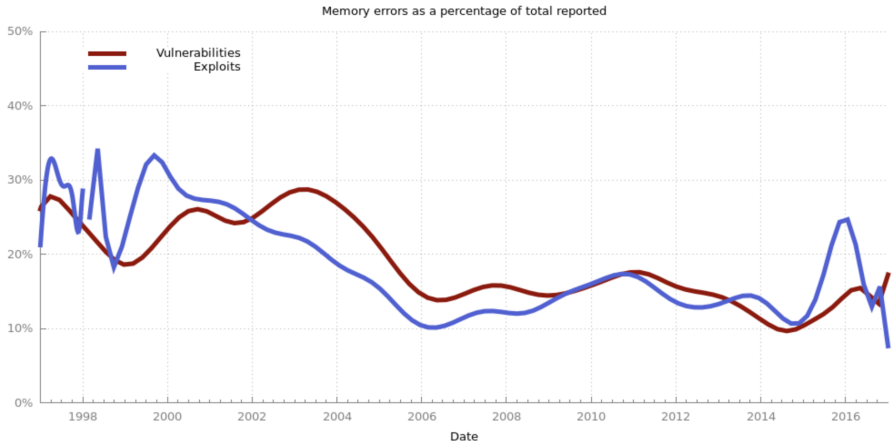

Figure 3.5: Relative number of memory-error vulnerabilities and exploits, subject to the total number of reported vulnerabilities and available exploits.

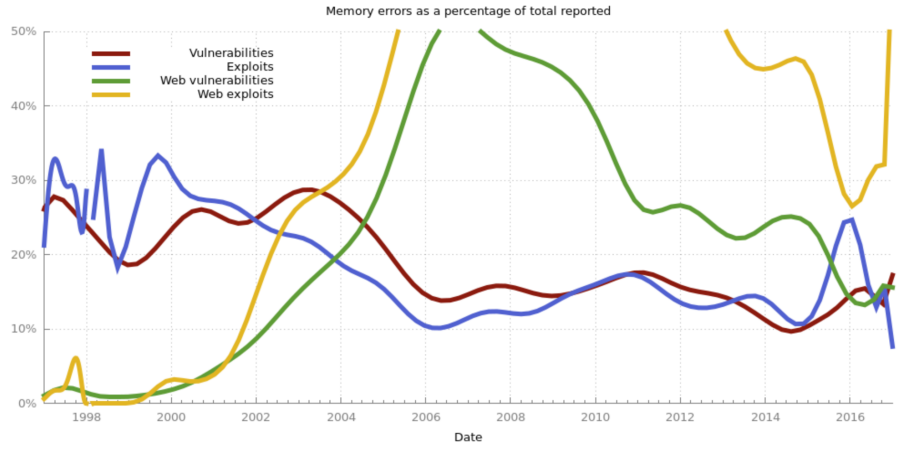

Figure 3.6: This figure shows the relative number of memory-error and web vulnerabilities and exploits, subject to their totals.

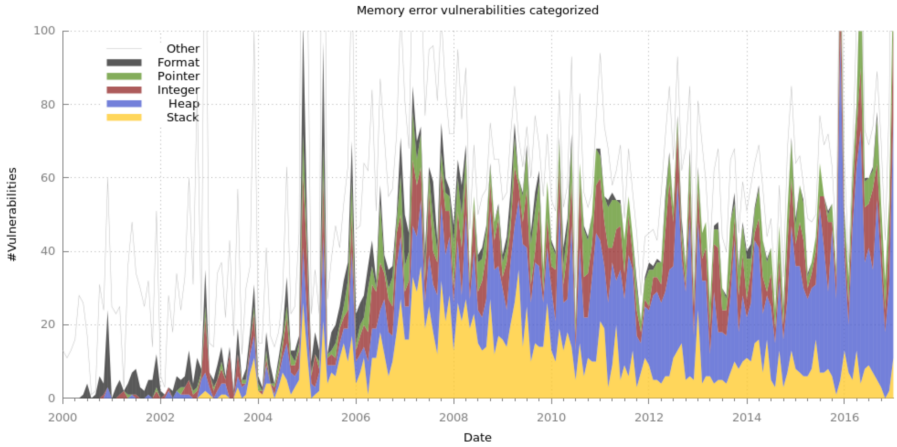

Figure 3.7: The number of reported memory-error vulnerabilities, categorized in stack-based, heap-based, integer, pointer, format string, and other issues.

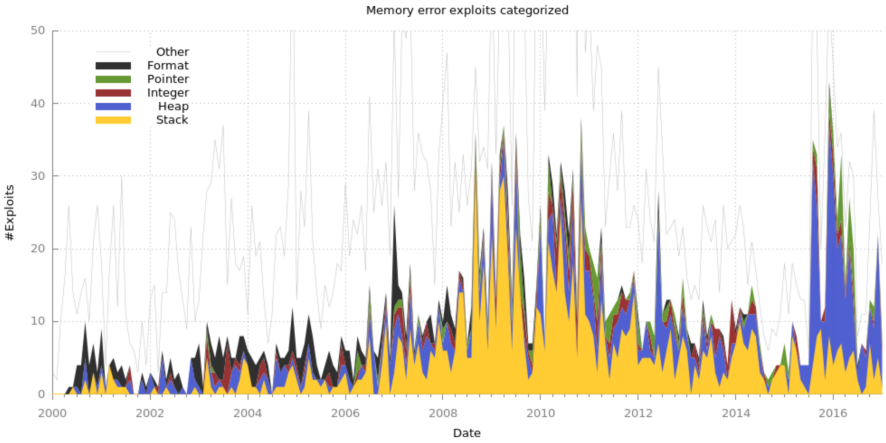

Figure 3.8: The number of reported memory-error exploits, categorized in stack-based, heap-based, integer, pointer, format string, and other issues.

3.6 Summary

The reports we analyze in this chapter provide useful and representative examples of the attack trends that we can experience today. In addition, all reports include predictions for the near future. Among the findings, this is the list of important and critical for *ReAct* observations.

- **New habits.** It is evident that new technologies create new attacker habits. A representative example is coin mining, which stands as a new attacker goal, much different than typical malicious activities, such as DDoS or Ad fraud. We stress here that all vendors have reported crypto mining as a *hot* problem, however there is little information of the techniques used to deliver coin mining. We can infer that con mining can be established using social engineering (tricking the user to click on something), web app exploitation (injecting malicious code in the web app) or by exploiting the browser. In *ReAct* we are interested in both web app exploitation by identifying and isolating the malicious web sites as well as defending the browser at the native level.

- **Ransowmware is on the rise.** Ransowmare and self-propagating malware is on the rise. All reports mentioned WannaCry and its variants as a serious threat. Moreover, ransomware was the only attack vector that was associated with very specific damage and financial loss (over 4 billions of USD). Based on the reports, we can infer that this is because (a) ransomware can compromise hosts without requiring user interaction, and (b) their actions are fairly severe. In *ReAct*, one of the key target is to defend the network from self-propagating malware that takes advantage of memory vulnerabilities.

- **Devices are not patched.** All reports mention the IoT landscape, and how several IoT devices pose a serious threat due to the absence of patching. One of the flagship techniques of *ReAct* is selective fortification, which aims at partial instrumenting a vulnerable host. This serves as a temporal patch, that can protect the vulnerable device from exploitation, before applying a real patch.

- **Passive attackers and watering holes.** Another new data point acquired by studying the reports is the existence of *watering holes*. This is compromised web sites, which aim at exploiting stealthy particular users and *not* massively all visitors. In *ReAct* we leverage techniques for identifying exploited hosts and isolating from the rest of the network.

- **Memory errors are still prevalent.** A key artifact of all this chapter is that memory errors are *still* prevalent, despite the evolution of

defenses and the evolution of our understanding. More importantly, it seems that fighting software exploitation based on memory errors is hard and, unfortunately, it stands as the attack class with the most severe consequences (see the discussion about WannaCry). In *ReAct* we invest significant past, present and future research in defending against this serious class of attack.

4 ReAct Threat Models

In this chapter, based on the definitions and examples of threat models we presented in Chapter 2 and the material of threat reports we studied in Chapter 3, we discuss all relevant threat models for *ReAct*. First, we discuss threat models, which stem primarily from incident reports as presented in Chapter 3. Then we give a short technical overview of *ReAct* just for projecting all relative tasks that need a threat-model definition. This short overview aims at making this deliverable self-contained with avoiding as much repetition as possible. Finally, we conclude with a set of very specific assumptions that are aligned with all the techniques designed, implemented and demonstrated in the context of *ReAct*. In short, the conclusion of this chapter conveys which attacks are interesting for *ReAct*.

4.1 Threat Models

ReAct is a framework that includes several different techniques. However, all techniques share a common *theme*, or, more technically, they focus on attacks that are aligned with a generic threat model. Of course, as we later expand, each technique exhibits different properties and, thus, different assumptions are in place.

We consider two threat models that are very aligned with the research activities of the project:

- **The Generic Threat Model.** This threat model defines the setting of the core technologies of *ReAct*, which are designed and implemented in WP3, WP4, and WP5.
- **The Advanced Threat Model.** This threat model defines the setting of additional research activities for next-generation attacks, which are explored in T2.1.

1. An attacker that can interact with a target vulnerable program by sending malicious inputs locally or over the network,
2. The attacker can leverage vulnerabilities to acquire an arbitrary write and read primitive,
3. The attacker wants to redirect the control flow of the program by overwriting control data using the write primitive,
4. The stack/heap is not executable and ASLR is in place,
5. The target program can be exploited and controlled by the attacker by utilizing the write/read primitives for launching ROP [27] or other more sophisticated code-reuse attacks [26].

Figure 4.1: *ReAct* generic threat model.

ReAct Generic Threat Model

The first, and core one, namely the *generic threat model* of *ReAct* which is close to what we discussed in Section 2.2.2. The primary reason for this decision stems from the following facts.

- **Exploiting systems, not humans.** Several severe incidents, as they where recently reported by major IT vendors, are based on exploiting system errors and not the human factor. For example, RansomWare (e.g., WannaCry) caused severe financial and operational damage in systems by leveraging purely, and only, vulnerabilities that can be found in popular software used by a broad range of users (from hospital infrastructures to just end users) and not by tricking users to install malware.
- **System software is still unsafe.** A large fraction of system software is still based on technologies that cannot prevent memory-based errors. For this reason, attackers find attractive exposing and using such vulnerabilities for taking control of systems. It is, therefore, important to develop *reactive* techniques for protecting our systems before attackers can launch their attacks.
- **Memory errors are still exploitable.** From a two-decade study of memory errors (presented in Chapter 3) we observe the memory errors are still popular and, more importantly, are still exploitable. No matter the defenses we have built in compilers or other software-

hardening protections, exploiting memory errors is still prevalent and can be recorded out in the wild.

Based on the aforementioned facts, the generic threat model that encapsulates all interesting attacks for *ReAct* is the one depicted in Figure 4.1. In Table 4.1 we provide a short description of how the properties (1)-(5) in the generic threat model were selected.

Property	Discussion
(1) An attacker that can interact with a target vulnerable program by sending malicious inputs locally or over the network.	Here, we assume both local and remote attackers. Local attackers have already access to a system, but they have limited capabilities (and they aim at privilege escalation), while remote attackers have no access to the system.
(2) The attacker can leverage vulnerabilities to acquire an arbitrary write and read primitive.	Here, we assume that the system *ReAct* protects may contain software vulnerabilities and these vulnerabilities can be strong enough to allow an attacker to read or write the memory of an executing process. Notice, that the Advanced Threat Model (discussed later) assumes that the system *does not* contain software vulnerabilities.
(3) The attacker wants to redirect the control flow of the program by overwriting control data using the write primitive	Here, we assume that the vulnerabilities can be actually exploited and let the attacker to change the legitimate control flow of a program.
(4) The stack/heap is not executable and ASLR is in place	Here, we assume that standard defenses are in place and the attacker does not launch attacks that are easy to contain (such as simply smashing the stack [3]).
(5) The target program can be exploited and controlled by the attacker by utilizing the write/read primitives for launching ROP [27] or other more sophisticated code-reuse attacks [26]	Here, we assume that the attacker can actually exploit the program using sophisticated attacks that are not easy to detect with current deployed techniques.

Table 4.1: Discussion of each of the five properties of the Generic Threat Model.

1. An attacker that can interact with a target program locally or over the network, without necessarily sending any inputs,
2. The target program is assumed to have *zero* software vulnerabilities,
3. The attacker can acquire a read primitive or write primitive by leveraging fundamental properties of hardware,
4. The attacker can exfiltrate secret information, using the read primitive, or corrupt memory (and then fall to the generic threat model of Figure 4.1), using the write primitive.

Figure 4.2: *ReAct* advanced threat model.

ReAct Advanced Threat Model

Additionally, **the second** threat model of *ReAct* encapsulates activities that will be primarily carried out in Task 2.2 (Next Generation Attacks). This threat model, which we name *advanced threat model*, has a more research base. Therefore, the plan of the project is to mostly explore attacks and defenses that are relevant, but *not* necessarily to build concrete systems that target the particular threat model. Nevertheless, we stress that this is a fairly advanced setting, as we discuss below, including attacker profiles and techniques that so far have *not* been reported in the real world. There have been zero incidents documented in the reports we include in this deliverable, and we are not aware of any real incident that involves such techniques, apart from academic research, of course.

We depict the advanced threat model in Figure 4.2. Notice, that compared to the generic one, this threat model includes a very powerful attacker. Essentially, the advanced threat model defines a setting where an attacker can effectively compromise a system by using *zero* software vulnerabilities. In other words, we assume a perfect world where software works as designed and there are no vulnerabilities that can lead to security incidents. In this setting, the attacker deliberately abuses certain hardware or operating-system features to either exfiltrate memory [19, 22], and thus leak important secrets, or corrupt memory [8, 31, 10], and thus transform the system to a new one, which can now be abused using the generic threat model.

Such attacks have only, so far, demonstrated in a research environment, they are very sophisticated, but *the lack* of typical assumptions (i.e., software vulnerabilities are not in place) make them very severe and alarming.

Although we have not seen, at this point, any of these attacks in the wild, it is important to explore the domain in advance. Therefore, the advanced threat model of *ReAct* defines the setting for the research of next-generation attacks, which are studied in T2.2.

In Table 4.1 we provide a short description of how the properties (1)-(4) in the advanced threat model were selected.

Property	Discussion
(1) An attacker that can interact with a target program locally or over the network, without necessarily sending any inputs.	Here, we assume both local and remote attackers. Local attackers have already access to a system, but they have limited capabilities (and they aim at privilege escalation), while remote attackers have no access to the system. Also, the attacker in this threat model can be simply passive.
(2) The target program is assumed to have *zero* software vulnerabilities.	Compared to the Generic Threat Model, here we assume that the system *does not* contain software vulnerabilities. Exploiting a system that does not contain software vulnerabilities is considered hard and attacks of this type have been so far explored in academic works.
(3) The attacker can acquire a read primitive or write primitive by leveraging fundamental properties of hardware.	Here, we assume that the attacker spins intrinsic properties of the system (not bugs) for acquiring primitives (read/write) that, so far, are acquired *only* by leveraging software vulnerabilities.
(4) The attacker can exfiltrate secret information, using the read primitive, or corrupt memory (and then fall to the generic threat model of Figure 4.1), using the write primitive.	Here, we assume that standard features of the system can be leveraged for putting a bug-free system to a state where the Generic Threat Model applies. Notice, that the system originally contains *zero* software vulnerabilities.

Table 4.2: Discussion of each of the four properties of the Advanced Threat Model.

4.2 Short Technical Overview

ReAct incorporates several *reactive defenses*. In short, *ReAct* classifies all reactive defenses based on the type of host they are applied to. Essentially, *ReAct* distinguishes hosts in a network based on being: (a) ordinary, (b)

vulnerable, (c) exploited, or (d) target of a future threat. Based on this taxonomy *ReAct* offers the following families of technologies:

- **Scanning/Probing** of ordinary hosts (WP3),
- **Forecasting** hosts that are targets of future threats (WP3).
- **Patching** of vulnerable hosts (WP4),
- **Isolation** of exploited hosts (WP4),
- **Forensic analysis** of exploited hosts (WP5).

Now, each of the above family of technologies includes several techniques implemented in a series of tasks and is associated with particular threats. For instance, when scanning/probing hosts we need to look for specific vulnerabilities, when patching hosts we fix, again, particular vulnerabilities, when isolating hosts we need to infer if the host was indeed exploited by, again, considering a set of relevant threat models, as well as when, finally, analyzing hosts for forecasting threats.

In the following part, we shortly review each core family of technologies and the associated threat models.

4.3 Specific Threat Models and Assumptions

Here we expand on the major components of *ReAct*. For each case, we briefly discuss some basic technical properties. For a full description of each of the components you need to refer to the *ReAct* DoW and successive deliverables.

4.3.1 Scanning and Probing

ReAct realizes *reactive* defenses. This means that the project aims to discover weaknesses first, and in particular, before the attacker manages to leverage weak parts of an organization. For this, one major component of *ReAct* is scanning the network and probing hosts for resolving vulnerabilities.

The major technical contribution of *ReAct* for scanning and probing is the actual techniques that are implemented in the relevant tools delivered by the project at a later phase. In this deliverable, we expand on the following question. *Which threat models* are relevant and leverage the *vulnerabilities* that *ReAct* scans/probes for? In other words, we are interesting in discussing what kind of *weak parts* are the ones that *ReAct* is searching for.

The vulnerabilities that *ReAct* is searching for is related to the generic threat model, which is outlined in Figure 4.1. In particular, *ReAct* utilizes a

vulnerability repository, and tests active services to infer if the services can be exploited. Such vulnerabilities have the following properties:

- They target native (unmanaged) software, which is usually written in C/C++ (although this can include other programming languages, as well),
- Vulnerabilities are related to memory errors,
- Memory errors can be both spatial, such as overflows, underflows, or type confusion, and temporal, such as use-after-free [30].

4.3.2 Forecasting

With this technology, we aim at bringing a proactive element to *ReAct*. By forecasting elements in the organizations that are most likely to get compromised by type of cyber threats explained in the generic threat model, we can prioritize the computers that will be scanned, patched and if needed isolated. Being very aligned with our generic threat model, according to the previous works on the cyber risk prediction domain, the most contributing factor to the cyber risk is existing vulnerabilities in software and the length of the vulnerability exposure on these computers or servers. Getting motivated from these findings, we can explore how much unknown vulnerabilities that are identified by the scanning component of *ReAct* contribute to the overall risk and how much patching them before attackers discover and use them might decrease it.

Although in our generic threat model, we do not focus on vulnerabilities due to the human error, the vulnerability lifecycle implicitly is affected by human behavior. Therefore, when we model our prediction technology, we will also exceptionally include features that are extracted from the user behavior.

In the course of this work package, we also aim at performing targeted forecasting. This way, we will not only provide predictions about generic threat infections but also the type of threats these entities might encounter. For example, the possible output of this component could be that a computer is likely to get infected by the ransomware threat [18], exploiting an existing vulnerability on the operating system or installed software on it.

Here we would like to also emphasise that cyber risk modeling and prediction are very challenging problems. To be able to advance prediction techniques that are adequately accurate, we cannot only focus on a particular threat model. For this reason, our prediction work will cover a wide range of threat models and will be dynamic to capture the changes on the threat landscape.

4.3.3 Patching

ReAct may infer that certain hosts are vulnerable through scanning and probing. As we stressed above, a vulnerable host for *ReAct* is the one that suffers from memory errors. The following major component of *ReAct* is to *temporary* repair the faulty hosts through *patching* [16, 15]. Nevertheless, no matter how the actual repairing (or patching) works, it is important to stress that the modified host will be protected *only* from attacks that are contained in the generic threat model (see Figure 4.1).

In practice, this means that, for *ReAct*, the vulnerable software will be instrumented in order to contain any exploit that triggers the particular vulnerabilities found. Notice, that this is not what is usually happening, today, through standard information-security operations. Once a bug is found, vendors attempt to eliminate the bug. *ReAct* follows a different approach. Instead of eliminating the bug, which sometimes may take time or introduce additional bugs accidentally, *ReAct* instruments the program for containing all exploits that leverage the bug. This means that after patching, the affected hosts can be *still* targeted in the context of the generic react threat model (see Figure 4.1), however, all attempts will be recorded and, in the worst case, result to crashing the host, instead of compromising it.

4.3.4 Isolation

The previously described defence mechanisms, such as patching, are applied *after* discovering that a host is vulnerable and *before* the host is compromised. On the contrary this defence mechanism (i.e., isolation) is applied *after* the host is compromised. Thus, to apply the defence approach of isolation we do not focus so much on the threat model (i.e. on the settings that led to the host compromising), but more on the compromising incident, itself.

ReAct may use a variety of mechanisms to infer that a host is compromised. Such mechanisms include (i) scanning, (ii) probing, (iii) passive network traffic monitoring, etc. All these sources of "signal" may be analysed and passed through an anomaly detection system that will conclude (with some degree of certainty) whether a host is compromised. For example, a host that makes an unusual high number of file accesses, balanced almost equally between read and write operations and happening completely outside normal back up times could be infected with ransomware. Once such hosts are identified, *ReAct* may choose to *isolate* these hosts. Such isolation may have multiple benefits: (i) it contains the infection within the host or at most within its local area network, (ii) it prevents the host from spreading the infection to other hosts (both inside and outside of the organization), and (iii) limits the damage that can be done to mapped network drives,

including drives at the cloud (such as Google Drive, OwnCloud, OneDrive, etc.). Such isolation may happen at two different levels:

- At the **host** level: if we have access to the compromised host we can remove it from the network and/or shut it down until it is free from any malware.
- At the **network** level: if access to the host is not possible, we can isolate the host by placing appropriate rules at the nearest firewall we can access. These rules will cut any communication between the host and the rest of the Internet until the host is again free of malware.

In *ReAct* we plan to employ a *modular* approach to detection and isolation. That is, *ReAct* will employ a detection framework where users can plug in their detection module. We plan to implement and demonstrate some example modules (possibly for Ransomware) and show how anomaly detection can be employed to detect compromised hosts. Our framework will be flexible enough to allow the plug in of future detection modules, possibly for malware that will appear some time in the future.

4.3.5 Forensic Analysis

In the unfortunate event of hosts getting exploited, *ReAct* contains the exploit, and therefore protects the rest of the network, by isolating the exploited hosts. Furthermore, *ReAct* incorporates techniques of advanced forensic analysis for identifying further malicious actions that took place on an exploited host. The forensic analysis offered by *ReAct* is *advanced* since potential targets have already been instrumented for emitting more sensitive and important data. This assists a forensic investigation by collecting additional fine-grained information about the data received by the component and the evolution of its internal state.

As in Isolation (WP4), during a forensic analysis the threat model might not be that interesting, since the attack has been already successful. Nevertheless, the forensic analysis of *ReAct* gives emphasis to incidents that stem from a successful exploitation attack based on the generic threat model ((see Figure 4.1).

4.4 Summary of Attack Scenarios

In this Chapter we have discussed the relevant threat models of *ReAct*. We summarize all work here.

The core threat model of *ReAct* is the *generic threat model* outlined in Figure 4.1. This threat model encapsulates advanced control-flow attacks, which can affect already hardened systems. *ReAct* assumes that state-of-the-art defenses are already in place, so all employed techniques aim to mitigate

state-of-the-art software exploitation (code-reuse and advanced code-reuse attacks).

To this end, here is a list of *five* possible attack scenarios. This least is not exhaustive but fairly representative.

- **Attack Scenario 1.** An organization is attacked by remote attackers that aim to exploit memory-error vulnerabilities that were so far unknown (zero-day),
- **Attack Scenario 2.** An organization is attacked by remote attackers that aim to exploit memory-error vulnerabilities that are known, but vulnerable software remains unpatched,
- **Attack Scenario 3.** An organization is attacked my local attackers that aim to exploit memory-error vulnerabilities (zero-day or known),
- **Attack Scenario 4.** An organization is attacked by already compromised machines that aim to further compromise additional hosts,
- **Attack Scenario 5.** An organization is attacked by currently hosted malware (downloaded accidentally by users).

All these attack scenarios can severely affect the operation of an organization, and they are variants of actual attacks that were reported by all IT vendors in their recent incident reports (see Chapter 3). For instance, WannaCry is one such case; it can attack an organization by exploiting memory-errors in unpatched Windows kernels.

We list a summary of all attack scenarios related to the four basic technologies of *ReAct* (scanning/probing, patching, isolating and forecasting) and assumptions discussed in Table 4.3.

Technology	**WP**	**Tasks**	**Attack Scenarios**
Scanning/Probing	WP3		In scanning/probing, hosts are checked for vulnerabilities in the context of the generic threat model (see Figure 4.1). Such vulnerabilities can be: • Spatial memory errors (buffer overflows/underflows, type confusion), • Temporal memory errors (use-after-free). The programs that are scanned/probed are all written in unsafe systems (C/C++) but can include sub-parts of other (safe) systems (Rust, Go, etc.).
Forecasting	WP3		Forecasting is the proactive element to *ReAct*. By forecasting elements in the organizations that are most likely to get compromised by type of cyber threats explained in the generic threat model, we can prioritize the computers that will be scanned, patched and if needed isolated. Compared to the other elements of *ReAct* (for scanning, patching and isolating hosts), which follow the generic threat model, forecasting includes the human error, since the vulnerability lifecycle implicitly is affected by human behavior. Therefore, when we model our prediction technology, we will also exceptionally include features that are extracted from the user behavior.

Patching	WP4		In patching, software is instrumented to contain very specific types of vulnerabilities. The vulnerabilities have been discovered by scanning/probing and they are all related to memory errors. *ReAct* does not repair the faulty software, but, instead, makes the vulnerability useless for exploitation. Again, the type of exploitation implied here is compatible with the generic threat model as outlined in Figure 4.1. This means that, once patching is done, the affected software can be still targeted, in the context of the generic threat model, but it cannot be exploited, since the software is selectively fortified for defending.

Isolation	WP4		*ReAct* may use a variety of mechanisms to infer that a host is compromised. Such mechanisms include (i) scanning, (ii) probing, (iii) passive network traffic monitoring, etc. All these sources of "signal" may be analysed and passed through an anomaly detection system that will conclude (with some degree of certainty) whether a host is compromised. For detecting a compromised host we do not focus on a particular threat model (i.e. on the settings that led to the host compromising), but more on the compromising incident, itself. Once *ReAct* infers that a host is compromised, attempts to isolate it. Such isolation may happen at two different levels: • At the **host** level: if we have access to the compromised host we can remove it from the network and/or shut it down until it is free from any malware. • At the **network** level: if access to the host is not possible, we can isolate the host by placing appropriate rules at the nearest firewall we can access. These rules will cut any communication between the host and the rest of the Internet until the host is again free of malware.

Forensic Analysis	WP5		In the unfortunate event of hosts getting exploited, *ReAct* contains the exploit, and therefore protects the rest of the network, by isolating the exploited hosts. During a forensic analysis the threat model might not be that interesting, since the attack has been already successful. Nevertheless, the forensic analysis of *ReAct* gives emphasis to incidents that stem from a successful exploitation attack based on the generic threat model ((see Figure 4.1).

Table 4.3: Summary of relevant threat models and assumptions to *ReAct*.

5 Conclusion

In this Deliverable, we list all threat models and related attack scenarios that are relevant to *ReAct*. This is done in the following four phases:

- We first discuss what *is* a threat model, and go through some very common threat models for making the reader more familiar (Section 2). In this discussion we include **two** threat models from network security and **one** threat model from software security.
- We then present a list of threat reports, created by major IT vendors in Information Security for identifying important real-world attacks that have been recorded recently (Section 3) and an ongoing study of the behaviour of memory errors over the last two decades [33]. The list of the reports contains **four** independently carried out studies from Symantec, Cisco, Akamai, and Europol.
- We then produce *generic* threat models that encapsulate a large fraction of real security incidents (Section 4) and are aligned with *ReAct*. In fact, we discuss **two** such threat models, namely the *ReAct Generic Threat Model* and the *ReAct Advanced Threat Model*. The first one, encapsulates activities in the core technical WPs (WP3, WP4, and WP5) and the second one is aligned with research for next-generation attacks, which is part of T2.2.
- We finally list **five** representative attack scenarios, where *ReAct* can apply reactive defenses. Additionally, we list all assumptions for each individual technique produced in the three technical work-packages of *ReAct*, namely WP3, WP4, and WP5 (Section 4).

Some *key* observations that stem from this deliverable and in particular from the studied reports are the following.

- Attackers may be interested in new malicious activities, which are aligned with the current technological evolution. As an example attackers may compromise servers for crypto-mining power, i.e., use the compromised hosts for mining digital coins, such as Bitcoin [20].
- Nevertheless, no matter the end goal, attackers *still* use memory errors to compromise machines. This happens, transparently, without social-engineering tricks.
- Despite the many defenses against software exploitation that is based on exploiting memory safety, we observe that (a) memory errors are still prevalent, (b) memory errors can be still exploited successfully. One *key* attack incident, as reported by all collected threat reports, is the success of ransomware [18] (such as WannaCry and Petya), which was delivered by exploiting memory errors [33].

Bibliography

[1] P. Akritidis, C. Cadar, C. Raiciu, M. Costa, and M. Castro. Preventing memory error exploits with WIT. In *IEEE Symposium on Security and Privacy*, pages 263–277. IEEE Computer Society, 2008.

[2] P. Akritidis, M. Costa, M. Castro, and S. Hand. Baggy bounds checking: An efficient and backwards-compatible defense against out-of-bounds errors. In *USENIX Security Symposium*, pages 51–66. USENIX Association, 2009.

[3] AlephOne. Smashing the stack for fun and profit, 1996.

[4] N. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. N. Schuldt. On the security of rc4 in TLS. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, pages 305–320, Washington, D.C., 2013. USENIX.

[5] R. J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley Publishing, 2 edition, 2008.

[6] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou. Understanding the mirai botnet. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1093–1110, Vancouver, BC, 2017. USENIX Association.

[7] S. Antonatos, P. Akritidis, V. T. Lam, and K. G. Anagnostakis. Puppetnets: Misusing web browsers as a distributed attack infrastructure. *ACM Trans. Inf. Syst. Secur.*, 12(2):12:1–12:38, Dec. 2008.

[8] E. Bosman, K. Razavi, H. Bos, and C. Giuffrida. Dedup est machina: Memory deduplication as an advanced exploitation vector. In *2016 IEEE Symposium on Security and Privacy (SP)*, volume 00, pages 987–1004, May 2016.

[9] X. Chen, A. Slowinska, D. Andriesse, H. Bos, and C. Giuffrida. Stackarmor: Comprehensive protection from stack-based memory error vulnerabilities for binaries. In *NDSS*. The Internet Society, 2015.

[10] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos. Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks. In *S&P*, May 2019.

[11] C. Cowan, M. Barringer, S. Beattie, G. Kroah-Hartman, M. Frantzen, and J. Lokier. Formatguard: Automatic protection from printf format string vulnerabilities. In *USENIX Security Symposium*, volume 91. Washington, DC, 2001.

[12] C. Cowan, C. Pu, D. Maier, H. Hintony, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, and Q. Zhang. Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks. In *Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7*, SSYM'98, pages 5–5, Berkeley, CA, USA, 1998. USENIX Association.

[13] T. H. Dang, P. Maniatis, and D. Wagner. The performance cost of shadow stacks and stack canaries. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, pages 555–566, 2015.

[14] R. Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. In *Proceedings of the 2005 Symposium on Usable Privacy and Security*, SOUPS '05, pages 77–88, New York, NY, USA, 2005. ACM.

[15] C. Giuffrida, C. Iorgulescu, G. Tamburrelli, and A. S. Tanenbaum. Automating live update for generic server programs. *IEEE Trans. Software Eng.*, 43(3):207–225, 2017.

[16] C. Giuffrida, A. Kuijsten, and A. S. Tanenbaum. Safe and automatic live update for operating systems. In V. Sarkar and R. Bodík, editors, *Architectural Support for Programming Languages and Operating Systems, ASPLOS '13, Houston, TX, USA - March 16 - 20, 2013*, pages 279–292. ACM, 2013.

[17] I. Haller, E. Göktas, E. Athanasopoulos, G. Portokalidis, and H. Bos. Shrinkwrap: Vtable protection without loose ends. In *ACSAC*, pages 341–350. ACM, 2015.

[18] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, and E. Kirda. Cutting the gordian knot: A look under the hood of ransomware attacks. In *DIMVA 2015, 12th Conference on Detection of Intrusions and Malware & Vulnerability Assessment, July 9-10, 2015, Milan, Italy*.

[19] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. Spectre attacks: Exploiting speculative execution. *CoRR*, abs/1801.01203, 2018.

[20] R. K. Konoth, E. Vineti, V. Moonsamy, M. Lindorfer, C. Kruegel, H. Bos, and G. Vigna. Minesweeper: An in-depth look into drive-by cryptocurrency mining and its defense. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 1714–1730, New York, NY, USA, 2018. ACM.

[21] J. Lettner, B. Kollenda, A. Homescu, P. Larsen, F. Schuster, L. Davi, A.-R. Sadeghi, T. Holz, and M. Franz. Subversive-c: Abusing and protecting dynamic message dispatch. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, pages 209–221, Denver, CO, 2016. USENIX Association.

[22] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018.*, pages 973–990, 2018.

[23] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, http://bitcoin.org/bitcoin.pdf.

[24] PaX Team. Address Space Layout Randomization (ASLR), 2003. http://pax.grsecurity.net/docs/aslr.txt.

[25] C. Rossow, D. Andriesse, T. Werner, B. Stone-Gross, D. Plohmann, C. J. Dietrich, and H. Bos. Sok: P2PWNED - modeling and evaluating the resilience of peer-to-peer botnets. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 97–111, 2013.

[26] F. Schuster, T. Tendyck, C. Liebchen, L. Davi, A.-R. Sadeghi, and T. Holz. Counterfeit object-oriented programming: On the difficulty of preventing code reuse attacks in c++ applications. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, SP '15, pages 745–762, Washington, DC, USA, 2015. IEEE Computer Society.

[27] H. Shacham. The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86). In *ACM Conference on Computer and Communications Security*, pages 552–561. ACM, 2007.

[28] U. Shankar, K. Talwar, J. S. Foster, and D. Wagner. Detecting format string vulnerabilities with type qualifiers. In *USENIX Security Symposium*, pages 201–220. Citeseer, 2001.

[29] K. Z. Snow, L. Davi, A. Dmitrienko, C. Liebchen, F. Monrose, and A.-R. Sadeghi. Just-In-Time Code Reuse: On the Effectiveness of Fine-Grained Address Space Layout Randomization. In *Proceedings of the 34th IEEE Symposium on Security and Privacy*, May 2013.

[30] L. Szekeres, M. Payer, T. Wei, and D. Song. Sok: Eternal war in memory. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 48–62, Washington, DC, USA, 2013. IEEE Computer Society.

[31] A. Tatar, R. K. Konoth, E. Athanasopoulos, C. Giuffrida, H. Bos, and K. Razavi. Throwhammer: Rowhammer attacks over the network and defenses. In *USENIX Annual Technical Conference*, pages 213–226. USENIX Association, 2018.

[32] C. Tice, T. Roeder, P. Collingbourne, S. Checkoway, U. Erlingsson, L. Lozano, and G. Pike. Enforcing forward-edge control-flow integrity in gcc & llvm. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14, pages 941–955, Berkeley, CA, USA, 2014. USENIX Association.

[33] V. van der Veen, N. dutt Sharma, L. Cavallaro, and H. Bos. Memory errors: The past, the present, and the future. In *Proceedings of the 15th International Conference on Research in Attacks, Intrusions, and Defenses*, RAID'12, pages 86–106, Berlin, Heidelberg, 2012. Springer-Verlag.

[34] Y. Zhou and X. Jiang. Dissecting android malware: Characterization and evolution. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 95–109, Washington, DC, USA, 2012. IEEE Computer Society.