Data Flooding against Ransomware: Concepts and Implementations

Davide Berardi[a], Saverio Giallorenzo[a,b], Andrea Melis[a], Simone Melloni[c], Loris Onori[a] and Marco Prandini[a]

[a] *Alma Mater Studiorum — Università di Bologna, Italy*
[b] *INRIA, France*
[c] *ARPAE Emilia-Romagna, Italy*

ARTICLE INFO

Keywords:
Ransomware
Detection
Mitigation
Restoration
Honeypot
Moving-target Defence
Resource Contention
Ranflood

ABSTRACT

Ransomware is one of the most infamous kinds of malware, particularly the "crypto" subclass, which encrypts users' files, asking for some monetary ransom in exchange for the decryption key. Recently, crypto-ransomware grew into a scourge for enterprises and governmental institutions. The most recent and impactful cases include an oil company in the US, an international Danish shipping company, and many hospitals and health departments in Europe. Attacks result in production lockdowns, shipping delays, and even risks to human lives.

To contrast ransomware attacks (crypto, in particular), we propose a family of solutions, called Data Flooding against Ransomware, tackling the main phases of detection, mitigation, and restoration, based on a mix of honeypots, resource contention, and moving target defence. These solutions hinge on detecting and contrasting the action of ransomware by flooding specific locations (e.g., the attack location, sensible folders, etc.) of the victim's disk with files. Besides the abstract definition of this family of solutions, we present an open-source tool that implements the mitigation and restoration phases, called Ranflood.

In particular, Ranflood supports three flooding strategies, apt for different attack scenarios. At its core, Ranflood buys time for the user to counteract the attack, e.g., to access an unresponsive, attacked server and shut it down manually. We benchmark the efficacy of Ranflood by performing a thorough evaluation over 6 crypto-ransomware (e.g., WannaCry, LockBit) for a total of 78 different attack scenarios, showing that Ranflood consistently lowers the amount of files lost to encryption.

1. Introduction

Liska and Gallo (2016) define ransomware as a "blanket term used to describe a class of malware that is used to digitally extort victims into payment of a specific fee".

A common kind of ransomware is of the *crypto* class, which holds hostage the files of the victim by encrypting them and then asking for a ransom for their decryption.

Background— In the last 10 years, the advent of new technologies changed the approach of ransomware (Greengard, 2021). Specifically, two innovations represented the turning point for the latest generation of ransomware: more efficient encryption mechanisms and the widespread adoption of cryptocurrencies. ~~Stronger encryption increased ransom-ware~~ More efficient encryption increased ransomware dangerousness both thanks to algorithms' speed, which shortened the useful timeframe that detectors have to trigger users and/or mitigations, and their strength, thwarting any attempts at reversing the process without a key. Cryptocurrencies provided criminals with reliable means to monetise attacks and protect their anonymity.

Just considering the last 5 years, we saw attacks becoming more and more frequent, with successful ones having strong side effects in global logistics, markets, and healthcare. *NotPetya*, which heavily targeted Ukraine in 2017 (taking offline some Chernobyl nuclear plant monitors (Griffin, 2017) and ministries, banks, and metro systems (Perlroth et al., 2017)), impacted at the global scale by blocking the logistics operations (and, thus, the hubs shared with other collaborators/competitors) of the Danish shipping company Maersk (Chappell and Dwyer, 2017), among many others. The attack, in 2021, ~~to~~ on the US Colonial Pipeline ~~company~~ companies caused fuel shortages in 5 states, leading to panic-buying, a surge in fuel prices, and fuelling disruptions (Joe et al., 2021). Attackers did not spare the health sector, which, since 2020, has been undergoing heavy pressures due to mass hospitalisation of COVID-19 cases and the management of national vaccination campaigns. Attacks have been ~~world-wide~~ worldwide — the heaviest happening in Ireland (Person and Padraic Halpin, 2021) and Italy (Abrams, 2021), similarly to the infamous WannaCry, which targeted in 2017 the UK healthcare system (Sheila A. and Tracy P., 2017) — and resulted in outages and delays of vital medical procedures.

~~*Contribution*—~~*Honeypot Techniques against Ransomware* ~~To contrast crypto-ransomware attacks, we propose~~ In this article, we focus on the usage of honeypot mechanisms for contrasting ransomware, and we introduce an advanced honeypot modality, which overcomes the limitations of current honeypot-based

davide.berardi6@unibo.it (D. Berardi); saverio.giallorenzo2@unibo.it (S. Giallorenzo); a.melis@unibo.it (A. Melis); smelloni@arpae.it (S. Melloni); loris.onori@studenti.unibo.it (L. Onori); marco.prandini@unibo.it (M. Prandini)

http://cs.unibo.it/~davide.berardi6 (D. Berardi); https://saveriogiallorenzo.com (S. Giallorenzo); https://www.unibo.it/sitoweb/a.melis/en (A. Melis); https://scholar.google.com/citations?user=F4-_jnwAAAAJ (S. Melloni); https://www.unibo.it/sitoweb/marco.prandini/en (M. Prandini)

ORCID(s): 0000-0002-2473-2439 (D. Berardi); 0000-0002-3658-6395 (S. Giallorenzo); 0000-0002-0101-2551 (A. Melis); 0000-0002-9535-8747 (S. Melloni); 0000-0002-3962-5513 (M. Prandini)

solutions.

In general, honeypots represent sacrificial resources that administrators use to either detect and/or ward off malicious intrusions. The idea is to provide easy-to-access decoy resources that, once accessed, expose the attacker and possibly slow it down.

We dedicate Section 3.1 to discuss in detail the limitations of existing honeypot techniques and Section 2 to provide a general review of the existing proposals. Briefly, basic honeypot techniques detect ransomware by deploying honeypot nodes, e.g., in the same network as those of real users, that contain decoy data. Advanced techniques (Moore, 2016; Al-rimy et al., 2018; Kok et al., 2019) omit using honeypot nodes and rather inject decoy files directly into real systems (e.g., the computers of the users). While these solutions increase the available detection surface (essentially, they make any node of a network a honeypot), they present problems linked to the pervasiveness of the honeypot files. For example, to cover the entire attack surface of a node one would need decoy files in all possible folders of that node and keep track of actions on all those files (Moore, 2016).

Contribution To overcome the limitations of existing honeypot techniques, we present a family of solutions based on a mix of *honeypots*, *resource contention*, and *moving target defence*. The underlying principle is that of flooding specific locations of the disk (e.g., the attack location, user folders, etc.) with ~~files. In Section 3 we show how this principle covers the~~ decoy files. Interestingly, our technique extends the coverage of honeypot mechanisms to the three main phases of ransomware contrast: detection, mitigation, and restoration. We call this new family of solutions Data flooding against Ransomware (DFaR). We dedicate Section 3 to introduce and discuss the concepts that characterise the DFaR approach.

~~Besides presenting DFaR from a theoretical point of view, we present~~ Then, we put into practice our theory by introducing an open-source tool, called Ranflood, which implements the mitigation and restoration phases of ~~the DFaRfamily~~DFaR.

At its core, Ranflood buys time for the user to counteract an ongoing attack, e.g., to access an unresponsive, attacked server and shut it down manually. ~~Detailing the aforementioned contrast techniques, Ranflood follows~~ In detail, Ranflood implements a *dynamic honeypot* approach, ~~that~~ which consists in generating decoy files and confusing the genuine files of the user with bait ones that the ransomware is lured into encrypting (making it waste time on them rather than on the actual files of the user). This confusion constitutes the moving-target-defence part of the approach. The third prong, that of resource contention, happens over IO access (e.g., for reading and writing on disk), which the ransomware must share with the (IO-heavy) Ranflood flooding routines.

The generation of (bait) files affords a wide design space spanning ~~over~~ different formats, structures~~and contents, which we start exploring in this work with three~~, and contents. In this article, we present three novel strategies, briefly introduced hereinafter and fully detailed in Section 4:

- *Random* generates files of different sizes and formats (those mostly targeted by ransomware (Lee et al., 2019)) with random content. The strategy has no prerequisites~~, once a target location is chosen~~besides the provision of a disk location to flood;
- *On-the-fly* performs a copy-based flooding using the actual files of the user. Besides requiring a target location, this strategy can entail a preliminary procedure (which shall run under ordinary situations, i.e. not during an attack) that collects lightweight file integrity information (e.g., checksum) of the user's files. This preliminary part is optional, but it can increase the effectiveness of the strategy by avoiding ~~to copy~~ copying files that have already been encrypted by ransomware;
- *Shadow* is also a kind of copy-based flooding strategy. Besides the target location, Shadow entails a necessary preliminary procedure that creates backups of the user's files—usually heavier than the integrity information collected by the On-the-Fly strategy—which it uses as the source for the copies. This strategy trades disk occupancy for increased effectiveness w.r.t. On-the-Fly, since all files ~~available before an attack are useful~~ in a backup are available for the flooding routine.

After presenting the general approach of Ranflood, its flooding strategies, and its software architecture in Section 4, we dedicate Section 5 to present a thorough benchmark of the efficacy of Ranflood. To perform this task, we consider 6 pieces of crypto-ransomware and measure the loss rate of user files (due to encryption) first without Ranflood and then using each of the three flooding strategies. Since the time-frame of execution can also be important, we simulate four incremental delays in the triggering of Ranflood, after the start of the ransomware. This amounts to 78 different scenarios. The results from Section 5 confirm our hypothesis: Ranflood consistently lowers the ~~amount~~ number of files lost to encryption.

While studying and investigating the approach we developed for Ranflood, we found interesting future research directions on detection, restoration, and on applications on kinds of ransomware other than crypto ones. We report these along with our concluding remarks in Section 6.

2. Related Work

Before presenting the contributions of this article, we discuss related work on the existing techniques for contrasting ransomware and relate these to our proposal.

Tracing an overview of the literature on anti-ransomware techniques means dealing with two main branches. The first ~~defines~~ regards work created specifically for a family of ransomware, while the ~~second, like~~ second—like the family of solutions presented ~~here, is~~ here—is of general application.

Within the first branch, we find mitigation techniques for the Cryptolocker ransomware. For example, Chew and Kumar (2019) presented a preventative technique based on altering access control levels of files and folders to revoke writing privileges during an attack. Lee et al. (2018) proposed a different approach, again targeting Cryptolocker, to recover from a ransomware attack by intercepting the decryption key of the ransomware either when the latter sends or receives it to/from its control server.

The other, larger branch of anti-ransomware solutions regards techniques that one can deploy regardless of a given family of ransomware.

For a general survey on (Windows-based) ransomware and the existing techniques for their detection and contrast, we point the reader to the thorough work recently published by Al-rimy et al. (2018), Kok et al. (2019), and Moussaileb et al. (2021). In the rest of this section, we focus on works that are the closest to ours. We organise the comparison with related work following the classification of the main phases of vulnerability management: detection, mitigation, and restoration.

We summarise our analysis in Table 1 to provide an overview and comparison of our proposal against the existing, related solutions, looked from the perspective of their contrast strategy, the phases ~~it applies to, its~~ they apply to, their coverage, and the ~~effort required from the user to apply~~ knowledge/effort that the solution requires for deploying/using it.

Detection— Detection schemes aim to identify ransomware attacks by monitoring specific activities. Some proposals use decoy files to detect ransomware. Moussaileb et al. (2018) use decoy folders and trigger a warning when a process passes through more than three of those folders. Moore (2016) proposed File Server Resource Manager (FSRM), a tool that triggers alerts when specific folders are modified in ways that are perceived as unusual w.r.t.~~the regular observed~~ the regularly-observed behaviour of the user. El-Kosairy and Azer (2018) worked on the placement of decoy folders to increase their likelihood of being the first victims of the ransomware, thus triggering a timely alert.

Scaife et al. (2016a) presented CryptoDrop, a tool that performs the detection of ransomware following three main principles—detect file format change, measure the change distance between files, measure the change of file entropy—and two secondary ones—detect file elimination and identification of a program that reads files of multiple formats but writes files in a single one. Another work in this category is HelDroid (Andronio et al., 2015), which works on mobile systems, and detects if an application attempts to lock or encrypt the device without the user's consent or if it displays some ransom request.

Kharaz et al. (2016) introduced a dynamic analysis system, called UNVEIL, based on the idea that, to mount a successful attack, ransomware must tamper with the user's files. UNVEIL automatically generates an artificial user environment able to monitor processes' interactions with user data and changes to the system's desktop as telltale signs of ransomware-like behaviour.

Other solutions, e.g., the ones surveyed by Kharraz et al. (2015), hinge on detecting and preventing (zero-day) ransomware attacks by looking at I/O requests and protecting the Master File Table (MFT) in the NTFS file system.

While the majority of proposals is host-based, network activity too can offer opportunities for ransomware detection. Recently, some solutions proposed to use Software Defined Networks (SDN) to detect ransomware. For example, Cabaj et al. (2018) proved that an SDN-based analysis of HTTP message sequences and of their respective content sizes can lead to detecting ransomware from the CryptoWall and Locky families. In a similar work, Akbanov et al. (2019) use OpenFlow (an enabler of SDN) traffic analysis to detect suspicious activities and to block infected hosts.

As seen here, honeypots are usually employed for detection. The approach of Data Flooding against Ransomware can be seen as a new, dynamic interpretation of honeypots that overcome the limitations of the existing approaches. We review these more in depth in Section 3.1, followed by a description of how detection works in our paradigm in Section 3.2.2.

Mitigation— Mitigation schemes strive to contrast the effects of ransomware attacks.

Works in this category frequently adopt some declination of the moving target technique (also part of the Data Flooding against Ransomware mitigation mechanism), e.g., "masking" user files, so that the ransomware skips them during the attack.

For example, Lee et al. (2019) analysed ransomware families and proposed a method that changes the extensions of files to formats normally skipped by ransomware.

Another example is Gómez-Hernández et al. (2018) where the authors proposed a general methodology called R-Locker to thwart crypto-ransomware actions. It is based on the deployment of honeypot archives, designed for the Linux system, to expose the ransomware when it accesses these. In addition to that, this approach can automatically launch steps to solve the infection.

This category hosts also OEM-provided solutions, e.g., Microsoft Windows 10 includes a "controlled folder access" feature (Microsoft, 2022), which works by allowing only trusted applications to access protected folders, configured by the user.

Here, the work closest to our tool for ransomware mitigation, Ranflood, is the one by Lee et al. (2019), since they both implement a moving target strategy. In addition to the latter, Ranflood deploys a resource contention countermeasure that further mitigates the action of the malware. The principle exploited by Microsoft's solution is of a different nature: it relies on user permissions to stop the action of a possible rogue program, but it does not prevent it from acting on any other, unprotected location.

Restoration— Restoration schemes concentrate on recovery the encrypted data after attacks.

An example of solutions in this category is ShieldFS (Continella et al., 2016), which relies on the integration between an ad-hoc file system and a detector (we list ShieldFS here since its main focus is recovery). When the detector recog-

Strategy	Detection	Mitigation	Restoration	Generic	Drop-in solution	Publications
Monitoring files	●	○	○	●	○	Scaife et al. (2016a) Andronio et al. (2015) Kharraz et al. (2015) Kharaz et al. (2016)
Key acquisition	●	○	○	○	○	Hassan (2019) Kolodenker et al. (2017)
Targeting files	●	○	○	●	●	Kharraz et al. (2015) Moussaileb et al. (2018) Moore (2016) El-Kosairy and Azer (2018)
Ransomware specific	●	○	○	○	○	Chew and Kumar (2019) Lee et al. (2018)
SDN traffic monitoring	●	○	○	●	○	Cabaj et al. (2018) Akbanov et al. (2019)
Restrict permissions	○	●	○	●	●	Microsoft (2022)
Extension randomisation	○	●	○	●	○	Evans et al. (2011) Lee et al. (2019)
Honeypot files	●	●	○	●	○	Gómez-Hernández et al. (2018)
Self-Healing file system	○	○	●	●	○	Continella et al. (2016)
Data flooding	●†	●	●‡	●	●	**This Work**

† not implemented in this article, ‡ copy-based flooding cf. Section 4

Table 1
Table comparing related works. Each row in the Table corresponds to a strategy found in one or more works related to ours—the last row corresponds to this article, for comparison—reported in the rightmost column. The other columns report properties of the strategy: to what actions it applies (detection, mitigation, restoration), whether it is generic (●) or specific (○) to a family of ransomware, and whether it is a drop-in solution (i.e., that only requires the user to install some software, as it happens e.g., for antiviruses).

nises a running ransomware, it activates a function of the file system that copies the data significant to the user to a location not reachable by the ransomware, for later restoration.

Also Ranflood, through its copy-based strategies (On-The-Fly and Shadow, cf. Section 4), provides a kind of recovery feature: if ~~some original file is~~ the original files are lost to the attack, the user has some chance to ~~find its content in one of~~ retrieve their content in the copies. One can refine this technique, e.g., by using the Shadow archive (if any) to restore files lost after the attack and by unifying replicas and offering post-attack file-recovery support (see Section 3.2.4).

While both ShieldFS and Ranflood are reactive recovery systems—that enact a response to an attack—the main difference with ShieldFS is that the latter is not a drop-in solution, since it entails switching to the namesake file system. ~~Ranflood's copy-based techniques require some preliminary configuration, but we deem this closer to configuring some software rather than formatting a whole drive~~

This comes with several disadvantages. First, the user needs to recompile the operating system kernel to correctly configure the ShieldFS solution. Second, being file-system-dependent, the solution is specific to the supported formats. Third, continuous porting between different versions of the same kernel is necessary to adapt ShieldFS to the latest version.

Contrarily to ShieldFS, the solution we propose is generic—this is witnessed also by the implementation of Ranflood (cf. Section 4), which uses the Java Virtual Machine for portability on any system that supports it—and requires only some preliminary configuration—similar to mainstream drop-in software applications.

like antiviruses.

3. Data Flooding against Ransomware

Before presenting relevant details of Ranflood, we introduce the family of techniques, called Data Flooding against Ransomware (DFaR), where Ranflood comes from—hinged on the dynamic honeypot approach. We start by positioning DFaR against the existing work on honeypots used to contrast ransomware. Then, we discuss how DFaR represents a family of techniques which includes applications to three main areas of vulnerability management: detection, mitigation, and restoration.

3.1. Dynamic ~~honeypots~~ Honeypots and Data Flooding against Ransomware

The essence of honeypots relies on the renowned scheme where administrators deploy easy-to-access computer resources that emulate the real ones present within the same network. These dummy resources must look as indistinguishable from ~~to~~ the actual ones as possible to an external intruder. Administrators isolate these resources from the real system to detect and slow down intrusions, setting up monitors to notify any suspicious activity (which is illicit by definition, since there is no reason for legitimate users to access the honeypot).

Previous works analysed the use of honeypots to detect ransomware (Moore, 2016; Al-rimy et al., 2018; Kok et al., 2019). The simplest declination of this approach lies ~~on~~ in deploying one or more honeypot nodes that contain data profiles similar to the ones attacked by ransomware. Then, monitors on the honeypot nodes can detect any changes to these static, isolated files and warn the administrators of the presence of ~~the~~ malware in the network.

More advanced techniques rely on using honeypots directly on the real nodes. The core of these solutions is to create honeypot folders and monitor them for changes. While the idea seems promising—essentially, making any node of the network a possible honeypot monitor for ransomware—the analysis performed by Moore (2016) on the existing techniques revealed a strong limitation to the approach. The problem, here, is that these solutions rely on static files always present on the disk of the user. Since the honeypot files can mix with the actual ones of the user, a solution that implements this technique must balance between its available trapping surface and the encumbrance it causes to the users. ~~Simply put, if the detection software created some honeypot files in locations frequently browsed by the user (usually, the ones mainly attacked by ransomware (Rossow et al., 2012; Y. Connolly and Wall, 2019; Continella et al., 2016)), e. g., the "Desktop" and "Documents" folders , the user could have interfering reactions upon discovering these "synthetic" files. For example, they could be alarmed and report false attacks to the administrators, or they could delete the synthetic data, trip the software's detector, and make it report false positives. Hence~~In essence, if one wanted to have complete monitoring of a whole machine, there should be at least one honeypot file in each of its folders. However, this quickly becomes inconvenient when mixing honeypot files with users' data. Indeed, users create, move, and delete folders in their ordinary work routines and they could trip the alarm of the detector. One could think of excluding these frequently used folders, but it would be a strong limitation of the range of the detector, ~~these~~ since most ransomware attacks those locations (Rossow et a which hold content sensitive to the user. Hence, honeypot solutions resort to using seldom-browsed (and attacked) locations and folders, thus limiting their trapping surface and strongly restraining their detecting ability: in the words of Moore (2016) "there is no way to influence the malware to access the area containing the monitored files".

The idea behind Data Flooding against Ransoware develops this take on ubiquitous honeypots against ransomware and gives it a Muhammad-and-the-Montain kind of twist:

> *if the ransomware will not come to the trap,*
> *then the trap must go to the ransomware*

Instead of using static files and incurring in the related trap-surface limitations, our intuition is to adopt a dynamic approach, where detection works by monitoring the activity of processes and by generating "floods" of honeypot files. If the process under inspection modifies the honeypot files—refined instantiations can analyse the patterns of data transformation to minimise false positives—we have strong evidence that it is some malware trying to lock the files of the user.

Working on the above idea, we found that one can use data flooding not only to detect ransomware, but also to contrast their action by mitigating their attacks and recovering from these.

The essence of the approach behind Data Flooding against Ransomware (DFaR) is to generate a deluge of honeypot files on demand in sensible locations, such as where the ransomware is executing or user folders, to detect and contrast the attacks. DFaR detection overcomes the limitations of existing honeypot solutions by adopting a dynamic stance towards decoy file deployment and their monitoring. DFaR mitigation (i.e., the contrast of an ongoing attack) has two benefits. On the one hand, it generates *resource contention* (Hunger et al., 2015) with the ransomware: its I/O operations compete on accessing the disk against the many ones induced by the flooder, slowing down the action of the former; on the other hand, data flooding performs a *moving target defence* action (Evans et al., 2011): the legit files of the users mix with the many decoy ones generated by the flooder, leading the ransomware to spend time (and I/O access) harmlessly working on honeypot files rather than on the sensitive ones. Recovery in DFaR can happen when mitigation used flooding techniques that generate files as copies of existing files of the user. Here, the idea is that, even if the ransomware encrypts the original copies of the user, we can recover the missing files using their pristine copies (if any).

3.2. Phases of Data Flooding against Ransomware

Before delving into the details of Ranflood—which implements an instance of the mitigation phase of DFaR—we

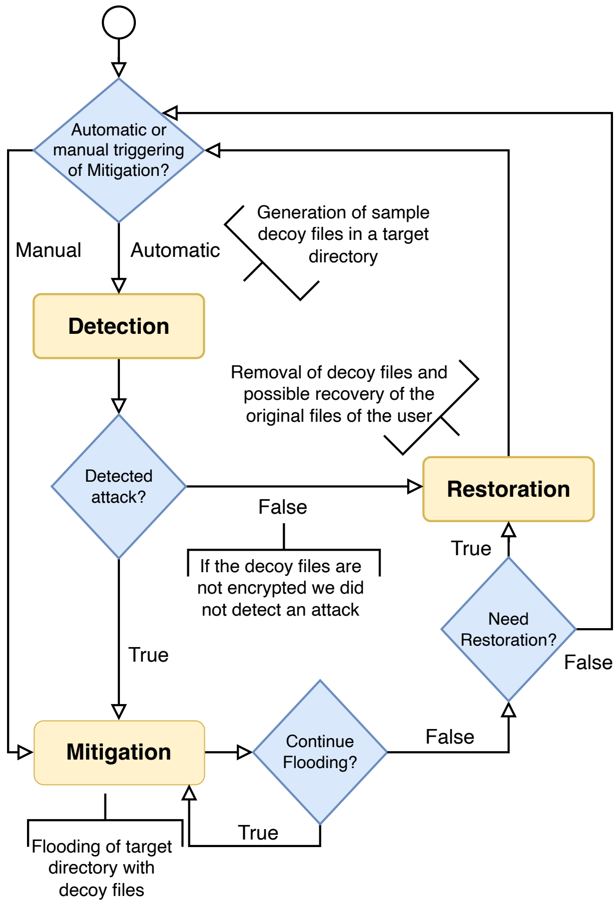

Figure 1: Flowchart of the relationship among the detection, mitigation, and restoration phases of Data Flooding against Ransomware.

focus on the main three phases that characterise vulnerability management through data flooding against ransomware: detection, mitigation, and restoration.

3.2.1. Three Phases of Data Flooding Against Ransomware

We report in Figure 1 a depiction of the relationship among the detection, mitigation, and restoration phases of Data Flooding against Ransomware. In the figure, we start (the top-most element) ~~from asking~~ with a choice which asks whether we want to follow the automatic or manual triggering of the mitigation phase. ~~In the first case, we use the detection mechanism of DFaRto trigger the launch of the Mitigation phase~~[1]~~. As mentioned~~As depicted in Figure 1, the Manual and Automatic activation modalities are mutually exclusive. The automatic activation implies the usage of a detector component that is able to identify the presence of an ongoing attack and triggers the mitigation phase.

The detection behaviour represented in Figure 1 is specific of DFaR. This is evident both reading the callouts that explain the behaviour of the elements and the relationship that the detection has with the restoration. However, in principle, one can use other, non-DFaR-based detection techniques (e.g., some of those reviewed in Section 2) to trigger the mitigation phase. In those cases, the detection would not necessarily interact with the restoration.

Looking at Figure 1, DFaR-based detection works by generating decoy files given a target location. Ideally, the detector would consider a time-window within which it expects the decoy files to be encrypted. If this happens, the detector trips an alarm (and possibly triggers the mitigation phase), otherwise the detector enters the restoration phase, which restores the original state of the target location as before the triggering of the detection, i.e., it safely removes the generated decoy files. When the mitigation phase starts, either triggered manually or by an automatic detector, it floods one or more target folders (e.g., where the ransomware is attacking, but also critical locations, independently of where the attack is running, such as personal folders of the user). This happens until the emission of a signal to stop the flooding (represented by the "Continue Flooding?" decision in Figure 1). After the mitigation phase, one can decide to run a restoration routine that removes the flooding files. Depending on the flooding technique employed, this phase can also restore the files of the user that might have been encrypted by ransomware.

We dedicate the remainder of this section to ~~provide~~ providing further details on how we envision the implementation of these three phases.

3.2.2. Detection

Regarding the practice of detection, we distinguish two modalities for the implementation of the detection phase, which hinges on how one defines the target location of the detection—i.e., where the detector deploys its decoy files.

The *static* modality is a mix between the traditional way of using honeypot files for ransomware and the novel dynamic take we present in this ~~paper~~article. In this case, the user defines a set of target locations that the detector periodically floods to spot possible ongoing attacks. This happens by having the detector perform what we call "mini-floods": it generates sets of random files in the target location(s) and monitors any activities on those files. If a program modifies said generated files in a way compatible with a ransomware (e.g., by replacing them with encrypted copies), then we have strong evidence that the suspect is indeed ransomware, against which we can launch the mitigation phase (e.g., Ranflood).

This modality partially overcomes the limitations of the traditional way of using honeypot files to detect ransomware. Indeed, ~~as mentioned in Section 3.1,~~ classic honeypot techniques for ransomware detection have the limitation of targeting seldom-used folders to minimise interactions with the user (that can result in false positives). On the contrary, the dynamic loop of flood-based detection (deploy files, monitor within a time-window, restore) makes it easier to monitor more trafficked, and more ~~likely to be attacked,~~ likely-to-be-attacked locations (such as the Desktop folder of the user).

[1] ~~Of course, one can combine other detection techniques to trigger the mitigation phase, such as the one reviewed in Section 2.~~

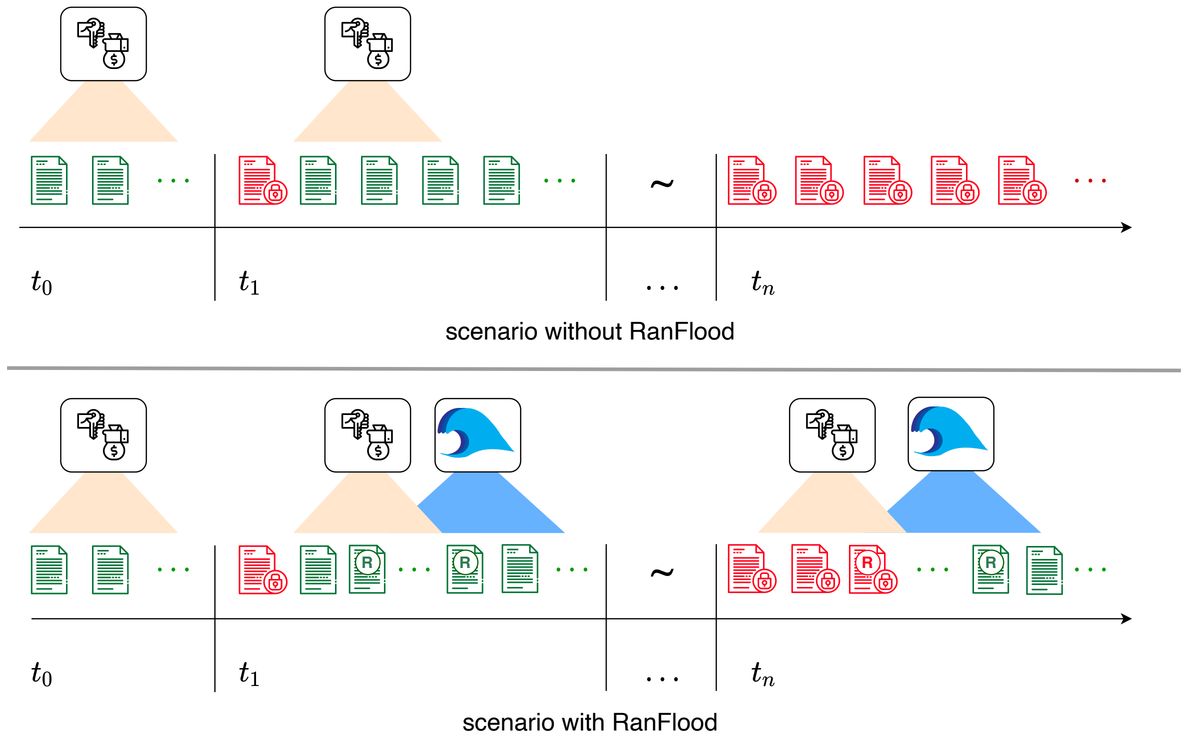

Figure 2: Depictions of the action of a crypto-ransomware (top) and the interaction between a DFaR-based mitigation tool (viz. Ranflood) and a crypto-ransomware (bottom).

~~The other~~ Alternative to the *static* modality is the *dynamic* ~~one~~modality. In this case, we envision a complementary process that "patrols" the system and triggers the detector on a specific set of locations. An example of one such patroller is a process that monitors the activities of the other running processes to spot behaviours that align with the execution profile of ransomware. In this case, the flood-based detector complements the activity of the patroller by dissipating the uncertainty of its detection logic, testing the hypothesis that the suspicious process is ransomware.

Of course, the design space of the patrolling process is quite wide, since it does not necessarily need to follow the flooding ~~approach — we~~ approach—we actually advise against using it as a patrolling routine, to avoid incurring ~~in~~ the limitations reported by Moore (2016) and discussed for the static ~~modality — but~~ modality—but can rather use complementary technologies such as process and file monitoring (Mehnaz et al., 2018) and machine learning (Gharib and Ghorbani, 2017).

The dynamic modality is the one we consider the most advanced and refined, which minimises the problems of classical honeypot techniques for detecting ransomware.

3.2.3. Mitigation

The mitigation phase represents a reaction to an ongoing ransomware attack, which a DFaR-based tool counteracts by flooding target folders—such as where the ransomware is performing its attack but also, as a preventative measure, locations with files critical to the user—with decoy files. ~~As mentioned, the~~ The principle is to stall the attack by confounding the authentic files of the user with a multitude of decoy ones, which the malware would waste time on encrypting.

Since Ranflood builds on the principles of DFaR mitigation, we use the description of this phase to introduce the general behaviour of Ranflood and dedicate Section 4.1 and Section 4.2 to respectively detail the three flooding strategies we implemented in Ranflood and the salient points of its software architecture.

To aid our presentation, we depict in Figure 2 a scheme of the action of some representative ransomware (top) and its interaction with a DFaR-based mitigation tool (bottom)—in the picture, we represent this tool with the Ranflood logo.

In the top part of the Figure, at time t_0 (the left-most block on the line), the ransomware starts its attack on a target folder by encrypting the files therein (the green documents represent the authentic files of the user). At time t_1, the ransomware has encrypted some files (viz., the red icons with a lock badge) and continues its action on the next ones. At time t_n, the ransomware has terminated the attack, and encrypted all files.

At the bottom of Figure 2, we show how a DFaR-based tool—specifically, Ranflood—contrasts the action above. In the Figure, the tool appears only after some detection mechanism activated it (as discussed in Section 3.2.2), at t_1.

The detection phase can instruct the tool to act on a specific set of folders, where the ransomware is performing its attack. However, this mitigation technique can also work under the weaker assumption that the detector found an ongoing attack, without indicating where this is happening, but the user specified sensitive folders to defend against the ransomware (e.g., the “Home” folder, “Documents”, etc.), which the tool floods with files. We respectively call these ~~*activity—*~~*activity-* and *location-based* activation modalities, and we deem both of them valid.

Of course, the activity-based modality is the most focussed of the two, as it contrasts the action of the ransomware in the location where it is deploying its attack. When one cannot rely on a detector able to spot where the ransomware is acting, the location-based mode provides a way to (preemptively) ward sensitive folders. Concretely, we also use the location-based modality in Section 5 to simplify the evaluation process of Ranflood, since it is not affected by the possible flakiness of activity-based flooding—which can change the target location of the countermeasure over different runs.

In general, one can even decide to deploy both activity- and location-based countermeasures to increase the effectiveness of the mitigation. The conjecture, here, is that the mix would simultaneously contrast the attack of the ransomware where it is causing damage, and ~~flood~~ flooding the critical folders to the user in advance. Since this is an advanced composition of ~~the mentioned~~ those modalities, we leave the empirical study of the effectiveness of their combination as future work.

Back to Figure 2, upon activation, the mitigation tool generates honeypot files (the documents marked with the “R” badge). The assumption we make is that, by generating a number of copies significantly greater than the number of legit files, the ransomware will more likely spend time on the former than on the latter. The ongoing action at t_n represents the mitigation effect of the tool, which hinders the attack of the ransomware and buys time for the users/administrators to intervene.

3.2.4. Restoration

After understanding how the detection and mitigation phases of DFaR work, one might wonder:

> *“Once we stopped the flooding of files, how do we restore the system as close as possible to the original state?”*

A possible answer to this question is what we dub the *outflow*, i.e., a restoration procedure tailored for DFaR-based detectors and mitigation tools. The principle backing this phase is the ability to discriminate between authentic and decoy files, to safely and effectively remove the latter.

When we consider flooding with decoy files filled with random content, restoration is a simple mark-and-sweep kind of task. However, this becomes an additional design dimension when paired with copy-based flooding modalities—where the decoy files are copies of the original files of the user; examples of these modalities are the On-The-Fly and the Shadow flooding modalities of Ranflood (presented in Section 4.1).

Indeed, in cases where we performed the flooding with copies of the original files, it can happen that the decoy files are the only valid copies of the original ones, of which we want to preserve one and use it in place of the lost original. In this case, one can define an outflow routine able to recognise when the authentic files of the user have been compromised and, if pristine copies of these are available as decoy files, use these to restore the former.

As expected, the implementation of the file-discrimination logic behind the outflow phase has a many alternatives. A naïve solution can rely on storing (preferably in a remote, safe location) the list of generated files, which we can later provide to the outflow. This is the logic implemented by the DFaR restoration tool (called “Filechecker”) we employ in our experiments in Section 5 to measure the effectiveness of Ranflood.

More advanced techniques can rely on digital fingerprinting (Stinson and Paterson, 2018, Chapter 13) to mark the flooding files in a way that prevents ransomware from performing quick analyses to detect a common signature and exclude them from its action. The idea, here, is to avoid saving any information on the fingerprinting process (e.g., the position of the fingerprints in the files) but rather rely on expensive fingerprint-inference procedures that statistically analyse the files and reconstruct the list of the generated ones[1]~~.~~.[1] Besides working as a watermarking procedure, we can use fingerprinting to hide some additional flooding information in the generated files. For example, for the file-copying flooding modalities, one can include in the generated files the path of the original copy, to help automatising the comparison-and-replacement process on the encrypted sources.

As a closing note on Data Flooding against Ransomware techniques, we highlight that these do not have particularly demanding prerequisites or dependencies (as opposed to some techniques reviewed in Section 2, e.g., which require the user to format the disk using a dedicated file system), and they work with the traditional file-access APIs provided by common operating systems. This positive trait makes DFaR-based tools (such as Ranflood) drop-in solutions, akin to the regular antiviruses users and administrators install on home and work computers.

4. Ranflood

We now focus our presentation on the relevant implementation details of Ranflood. Namely, we present the three novel flooding strategies that Ranflood provides and its software architecture.

[1] ~~To harden the task for the ransomware, one can use sets of fingerprints, which forces the ransomware to either spend time on piecemeal inference computations or give up.~~

[1] To harden the task for the ransomware, one can use sets of fingerprints, which forces the ransomware to either spend time on piecemeal inference computations or give up.

4.1. Three Data Flooding Strategies

To streamline the presentation of the three flooding strategies we designed and implemented in Ranflood, we delineate these via simplified pseudo-code, useful to pinpoint their qualitative differences, pros, and cons. We provide more details on their actual, more sophisticated implementation in Section 4.2.

4.1.1. Random

Nomen omen, the Random flooding strategy, sketched in Algorithm 1, floods a given location (*path*, in the pseudo-code) with randomly-generated files. It incarnates the basic form of flood-based mitigation: slowing down the ransomware via resource contention and moving-target defence. The strategy has the smallest friction to its deployment among the three we are presenting, as it does not entail pre-flooding configurations by the user (as discussed for the On-The-Fly and the Shadow strategies, below).

Algorithm 1: Random Data Flooding

```
input: path, minSize, maxSize
FILE_EXT ← [".doc",".pdf",".xls",".jpg",".mp4",..];
while keepFlooding do
    f_size ← randomInt(minSize,maxSize);
    cnt ← newByteArray( f_size);
    ext ← rndSelect( FILE_EXT);
    append( cnt, getHeader( ext));
    seed ← random64Seed() ; // 64-bit number
    for i ← 0 to i < ( capacity( cnt) / 64 ) do
        seed ← seed ^ ( seed ≪ 13 );
        seed ← seed ^ ( seed ⋙ 7 );
        seed ← seed ^ ( seed ≪ 17 );
        append( cnt, seed);
    end
    if capacity ( cnt ) > 0 then
        r ← newByteArray( capacity( cnt));
        r ← fillWithRandomBytes( r);
        append ( cnt, r );
    end
    writeFile( rndFilePath(path, ext), cnt);
end
```

We expect ~~an~~ the implementation of the strategy to be effective if it meets three conditions: (1) it generates files using extensions that ransomware usually target (Rossow et al., 2012; Y. Connolly and Wall, 2019; Continella et al., 2016) (e.g., in Algorithm 1, and in Ranflood, we use common formats such as ".pdf" and ".jpg"); (2) the generated content of the files does not give way to analyses that let the malware suspect of their synthetic nature (e.g., reusing the same sequences over and over or having file headers that do not match the standard format of their related extension); (3) it produces large ~~amount~~ amounts of such files in a short time-frame.

The code in Algorithm 1 achieves (1), (2), and (3) to a satisfying degree. In particular, we deem (2) and (3) of good level for two reasons. One, because we use a variant of Xor-shift (Marsaglia, 2003) for fast randomness (the first **for** loop in Algorithm 1) to quickly generate random content for files of random sizes—in the $[minSize, maxSize]$ interval, e.g., Ranflood uses ~~common~~ file sizes in the range $minSize = 2^8$ and $maxSize = 2^{22}$ as default values, but the user can also configure these. Moreover, we make the format of the file (declared by its extension) and its header match—the first instruction that `appends` to the *cnt* array the byte sequence related to its *ext*ension (`getHeader`). The `rndFilePath` function generates a random file path (location, file name) under the given *path* and with the given *ext*ension.

4.1.2. On-The-Fly

The On-The-Fly flooding strategy is the first we present that performs a copy-based flooding. Essentially, we replace the generation of synthetic files performed by the Random strategy with the generation of copies of actual files found at a flooding location. File replication adds a layer of defence to the Random strategy, as it helps to increase the likelihood of preserving the users' files by generating additional, valid copies that might escape the ransomware.

Not all files have equal importance for this strategy. The basic rule we introduce, here, is skipping the replication of encrypted files, since they worsen the performance of the strategy; copying these files is detrimental in two ways: a) it wastes the time of the flooder on files useless to the user and b) it generates files that the malware would skip, recognising them as already encrypted.

The solution we develop to tackle this issue is to add a preliminary "~~snapshotting~~ snapshooting phase" to save a list of the valid files, later used during flooding for efficient discrimination. Saving such a list trades a small occupation footprint on the disk with an increase in the efficacy of the flooding.

Specifically, the ~~snapshotting~~ snapshooting procedure reported in Algorithm 2 saves a `digest` (e.g., MD5) of the content of the user files and uses it as an integrity verification code to validate the files during the flooding phase (Algorithm 3).

For simplicity, in Algorithm 3, at each iteration we read (`readBytes`) the files from disk and write (`copy`) them, if valid. While this could be a reasonable implementation, it leaves open the possibility to lose files between iterations[2]~~.~~.[2] To avert this risk, Ranflood runs a more sophisticated version of Algorithm 3, not shown here for the sake of clarity, that caches the content of the files read once from the disk and then iterates their replication (trading memory occupation for effectiveness).

We close the description of On-The-Fly noting a subtle detail: saving snapshot lists exposes the strategy to failure due to the action of the ransomware, which could encrypt

[2] ~~Imagine, in the first iteration, that we replicate the valid file f in f', the ransomware encrypts both of them, and we lose (the possibility of copying) the content of f.~~

[2] Imagine, in the first iteration, that we replicate the valid file f in f', the ransomware encrypts both of them, and we lose (the possibility of copying) the content of f.

the list itself. This is a general problem of any software that uses secondary memory for its functionality (e.g., for configuration, runtime, etc.) and one can mitigate it a) via remote file storage, like NAS and the Cloud, and b) using locations and (random, exotic) file extensions for lists, which ransomware usually skip. We omit to discuss this problem here and plan to address the subject in future extensions.

Algorithm 2: On-the-fly ~~Snapshotting~~Snapshooting.

```
input: path
for file in walkFiles ( path ) do
    if isFile( f ) then
        saveOTFSnapshot( path, f,
            digest( readBytes( path, f )));
    end
end
```

Algorithm 3: On-the-fly Data Flooding

```
input: path
while keepFlooding do
    for f in walkFiles ( path ) do
        b ← readBytes( path, f );
        if getOTFSnaphot(path, f) = digest( b) then
            copy( b, randomFilePath(path));
        end
    end
end
```

4.1.3. Shadow

The Shadow strategy is a variant of the On-The-Fly one (indeed, Algorithms 4 and 5 of Shadow are close to, respectively, Algorithms 2 and 3 of On-The-Fly), where snapshots save the full content of the files of the user rather than more lightweight information, such as their fingerprint.

Since the Shadow ~~snapshotting~~ snapshooting phase follows the traditional process of backup systems, it also suffers the same, known trade-offs of local, on-site, and remote backup storage/retrieval. In Ranflood, we use (tar.gz) archives to try to minimise the space required for snapshots and preserve those archives on the same disk of the original copies, both for simplicity and to minimise loading times. More advanced implementations could use secondary disks, NAS, and the Cloud to mitigate the possibility of losing the local backups, if targeted by the ransomware.

Algorithm 4: Shadow Snapshooting.

```
input: path
for file in walkFiles ( path ) do
    if isFile( f ) then
        saveShadowSnapshot( path, readBytes( f ));
    end
end
```

Algorithm 5: Shadow Data Flooding

```
input: path
while keepFlooding do
    for cnt in getShadowSnapshots ( path ) do
        writeFile( rndFilePath(path), cnt);
    end
end
```

4.2. Software Architecture

~~As mentioned, the~~ The implementation of the strategies from Section 4.1 in Ranflood are more sophisticated, technically complex, and tuned to exploit the maximal degree of concurrency available on the attacked node—maximising both IO access contention and the file generation rate. Hereinafter, we report on the salient elements of the software architecture of Ranflood that ~~support~~ supports this high degree of concurrency.

The Ranflood Architecture~~—~~ Two components determine the behaviour of Ranflood.

First, the *Ranflood engine* implements refined versions of the algorithms shown in Section 4.1. We call these elements *operations*, e.g., one operation can be an instance of the Random flooding strategy or the ~~snapshotting~~ snapshooting routine of the On-The-Fly strategy. While in Section 4.1 we represent strategies as indivisible units, in Ranflood one operation corresponds to ~~a number of~~ several executable tasks without *a priori* bounds, e.g., once we execute a Random flood operation, it generates an unlimited amount of tasks (until the user commands the termination of that operation) and each task carries the code for the generation of one, specific random file. Since we envision the Ranflood engine to manage multiple concurrent commands, possibly launched from different sources (e.g., the user, an automatic detector, etc.), we opted for a Client-Daemon model (Tanenbaum, 2009, Chapter 2). Specifically, the engine works as a daemon process in the background, not associated with a particular user, and users/programs interact with it with lightweight, asynchronous clients/interfaces.

The second component is the *task manager*, which handles the scheduling of operations and their tasks. Indeed, at runtime, we equate launched operations and their tasks as generic work that the task manager schedules for execution. The difference between an operation and a task is that the former generates other tasks, while the latter performs I/O interactions. Concretely, we implemented the task manager following the Proactor (Pyarali et al., 1997) event-handling pattern. The Proactor decouples the task demultiplexing and the task-handler scheduling logic from the actual behaviour enacted by the single tasks, ~~in an asynchronous way~~asynchronously. This execution method helps in further exploiting the parallelism available on the attacked node and in minimising the effect of I/O overhead and latency. Moreover, isolating tasks makes operations more resilient: if a task fails, it does not affect its operation or the other tasks.

~~Currently, Ranflood (~~We further clarify the architecture

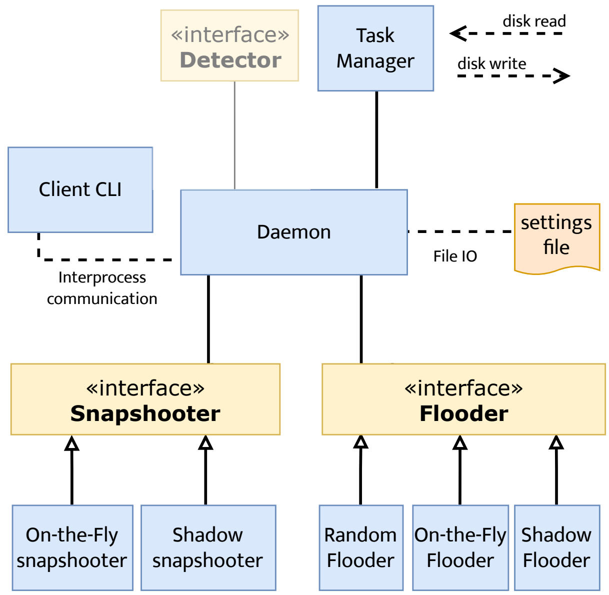

Figure 3: Model of Ranflood's Architecture.

of Ranflood by depicting a model of it in Figure 3. In the Figure, we highlight the (interprocess communication) interaction between the Client Command-line Interface (Client CLI) and the Daemon. Besides issuing commands for contrasting ransomware, the Client can also set configurations of the Daemon, which the latter stores in a settings file. The other main components are dedicated to implementing the different flooding strategies. The basic interface for the latter is Flooder, which the Random, On-the-Fly, and Shadow flooders implement. The On-the-Fly and Shadow strategies also have a snapshooting phase, which they realise by implementing the Snapshooter interface. The Daemon interacts with these components to obtain tasks that operate on files, ran in parallel by the Task Manager—which implements the Proactor's logic. The faded Detector interface indicates that the Ranflood Daemon is organised to support the integration of (generic, i.e., no necessarily DFaR-based) detectors.

Ranflood (both its client and daemon) ~~, at~~ (at the time of writing at version 0.5.9-beta~~,~~) is an open-source project[3] written in Java, uses the RxJava[4] library for the basic components of its task manager and, through the GraalVM[5] compiler, it is available as native binaries for Windows, macOS, and Linux systems, besides its Java executable.

5. Evaluation

We now present our evaluation ~~on~~ of the effectiveness of Ranflood in lowering the loss rate of files due to ransomware attacks. To perform a thorough evaluation, we test Ranflood under different conditions: we select 6 ransomware samples, we consider 4 increasing activation delays of Ranflood (which simulate in a deterministic way the triggering by a detector), and test each of its 3 flooding strategies. ~~This results in~~

The 4 increasing activation delays are important to investigate the relationship between the time it can take detection to activate Ranflood (i.e., to account for different timeframes for the automatic triggering of the mitigation, cf. Figure 1) and the effectiveness of the Ranflood action. In this article, we ditched the use of some specific detection technology to avoid introducting additional variables into our experiments—the most prominent of these being the variance in detection times. Hence, we take 4 fixed delays which represent increasing worst-case activation scenarios (we discuss the actual times in Section 5.1, which are inspired by studies from the literature). Future work can focus on studying the relationship between different families and implementations of detection techniques and Ranflood. To this aim, one would need to systematically review the literature on ransomware detection, select a set of representative families of detectors, select implementations for each of these families, and establish and run statistically-relevant batteries of benchmarks.

The combination of the ransomware samples, the activation delays, and the flooding strategies gives us 72 different run scenarios, totalling 78 considering also the 6 baseline runs where we do not let any Ranflood strategy run (called "None" configurations). We run each scenario 4 times, reporting the averages. Before showing the results, we detail the target operating system and data used in the tests, the selected piece of ransomware, and how we measure the loss rate in the tests.

5.1. Benchmarking Method

Target Operating System and Data— To select the target operating system, we choose to adopt the one with the wider market share on desktop machines in the last year (at the time of writing). To find it, we used the data made available by StatCounter[6], which reports a marked share of around 75% held by Microsoft Windows 10. Thus, we use this operating system as target.

The target data is the set of files attacked by ransomware. Since the ransomware samples we consider mainly attacks the profile of the user in the machine, our target data corresponds to a representative set of files of an ordinary user (Continella et al., 2016; Kharaz et al., 2016; Akbanov et al., 2019).

There are mainly two ways to obtain a profile of this type.

The first is organically, i.e., drawn from a real environment used by a regular user for a certain amount of time. Continella et al. (2016) and other authors (Kharaz et al., 2016; Akbanov et al., 2019) followed this approach, using in their tests the profiles of some users who worked on the test environment for e.g., a week. Two main drawbacks of this approach are: a) it might not generate a significant amount of data, since it depends on the type of activity of the user and the recording timeframe, and b) it requires precautions, e.g.,

[3] We will make the link to the repository available in case of acceptance of the article.

[4] https://github.com/ReactiveX/RxJava.

[5] https://www.graalvm.org/.

[6] https://gs.statcounter.com/os-market-share#monthly-201807-202111.

we need to make sure the data is anonymised, to avoid, e.g., spreading sensible information of the user. The second approach is to create the profile synthetically, but starting from real-world skeletons and populating them. Here, the drawback is that the generated data is not organic. On the positive side, we do not depend on some selection of users or some timeframe.

Since we choose to ditch using a detector, which would instruct Ranflood to act on the attack location of the ransomware, we just need to have an ordinary user profile skeleton and command Ranflood to ward/flood those sensible folders (the location-based activation modality discussed in Section 3.2.3). Hence, we deem it appropriate to follow the second approach and build a synthetic, but realistic target profile.

To do this, we built on the skeleton reported by Halsey (2016), who ~~defined the main~~ identified the main user paths and folders of the Windows 10 File System. Then, for the user files, we generated 2GB of data, following the indications of Kaspersky (2021) and Scaife et al. (2016b) on the formats most subject to ransomware attack. Besides the format, we also followed other guidelines to tune the profile for the task: we created files with names usually preferred by ransomware (Kroll, 2021; Anderson and McGrew, 2016) and, following the suggestions by (Rossow et al., 2012), we gave to the profile a user-interactivity imprint by installing a set of applications among the most used, like a browser and an office suite.

In the generated profile, we have 13 folders, among which "Documents", "Desktop", "Music", and "Pictures", which we consider sensible to the user and which we flood and monitor to calculate the loss rate after each attack.

Ransomware— To identify the ransomware samples for the tests, we used the VirusTotal Intelligence API to obtain the current Windows executables associated with the main ransomware families. We obtained a set of samples (including CryptoWall, TeslaCrypt, WannaCry, Certbot, NotPetya, and Critoni), which we tested to actually execute in our target environment. Not all samples worked, e.g., some samples did not receive instructions and public encryption keys from their control servers and did not perform any attack. We filtered out these samples, to only focus on active ones. Moreover, we excluded ransomware that forced the machine to restart. This is not a problem from the functional point of view of Ranflood (which we could instruct to start its routine after the reboot), but it would make the tests more unreliable, since we would not know any more the exact delay between the start of the ransomware and Ranflood. Thus, we also removed these samples. The resulting set of samples include 6 pieces of ransomware: GandCrab, LockBit, Phobos, Ryuk, Vipasana, and WannaCry.

Logs and Metrics— The final ingredients of our evaluation method are 1) the execution timeframe, i.e., how much time we let the ransomware and Ranflood execute and 2) the 4 activation delays of Ranflood, to simulate the triggering from a detector. For the timeframe, we ~~deemed it appropriate to set it to 10 minutes~~run preliminary experiments and saw that 10 minutes are generally appropriate to witness the full extent of a ransomware attack on users' folders—this is matched by results from other researchers who verified that the action timeframe of different families of ransomware is within 4 to 9 minutes (Zuhair and Selamat, 2019; Ahmed et al., 2020). For the delay, we consider detectors that respectively require the ransomware to run for 5%, 10%, 30%, and 50% of the timeframe before triggering Ranflood, hence 1/2, 1, 3, and 5 minutes. We selected these delays to look at the worst-case scenarios, starting from the high-end values of the detection time spectrum, ranging around 30–40 seconds (Zuhair and Selamat, 2019 , and looking at even less performant cases with the 1-, 3-, and 5-minute delays.

The data points we want to collect in the tests are two: the number of files lost to encryption and, for copy-based strategies, the number of files saved through copying (i.e., when we lost the original file but have a pristine copy). To compute this data, we let the piece of ransomware and Ranflood run for the length of the timeframe, we shut the test machine down, and then mount the disk on a different machine to analyse it (this is necessary to make sure that the piece of ransomware cannot modify the files any more). To calculate the data loss, we compare the digests of all the files in the target profile (collected beforehand) against the files in the mounted drive—we use this method to find all valid files, both the original and the replicas, counted once (i.e., all files with the same digest count as one).

5.2. Testbed

To run the tests, we assembled a testbed made of a cluster of test nodes with hardware representative of today's ordinary office/desktop personal computers. The tests nodes ran isolated Windows 10 virtual-machines, orchestrated by a central gateway running Ubuntu 21.04 (to further avoid possible interactions with ransomware samples in the cluster). The gateway of the testbed was the only terminal with network access (this avoided problems like the escape of some ransomware, e.g., due to unknown network exploits, and the execution of unexpected processes, e.g., update routines, which might interfere with the performance). Figure 4 reports a schema of the testbed, where "PVE" prefixes the test nodes. The main point of assembling this testbed was to automatise and standardise the tests and make our data as reliable as possible.

Regarding the nodes, we used four desktop computers each equipped with an Intel i3-4170 (3.70GHz) dual-core, ~~four-threads~~ four-thread CPUs, 12GB of RAM~~and a hard disk~~, and a Hard Disk Drive[7] (HDD) of 500GB. These machines run ProxMox version 7.0-8 on GNU/Linux. We built the template for the virtual machines from the one provided by Microsoft of Windows version 10 (x64) Stable 1809. Each

[7] Since IO contention is a fundamental element of the Ranflood contrast action, future empirical studies can extend the types of storage devices used for the testbed to other technologies like Solid-State Drives (SSD), Non-Volatile Memory Express (NVMe) drives.

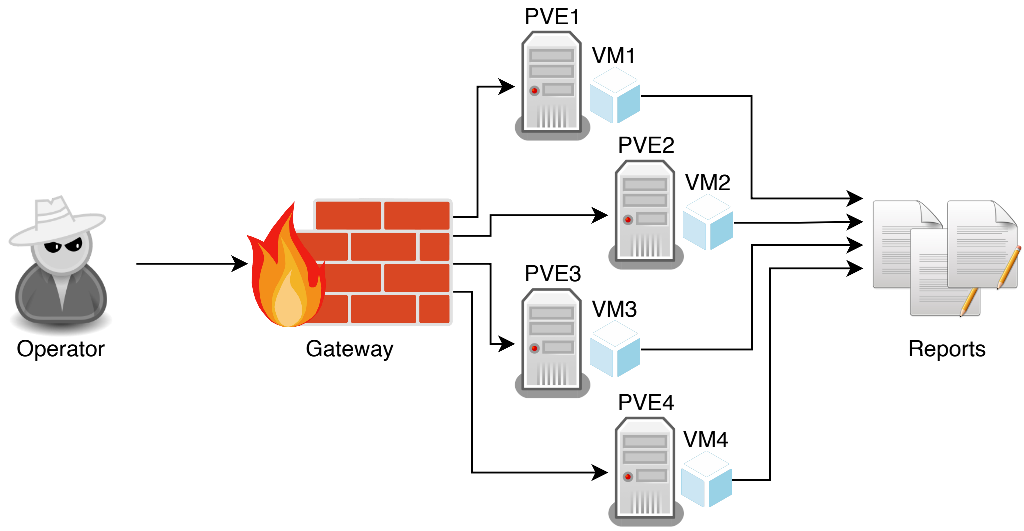

Figure 4: Testbed schema. The operator connects to the Gateway to run the tests and retrieve the reports. The test nodes (PVE*) host one virtual machine each.

node runs one virtual machine with a dual-core, four-thread CPU, 12GB of RAM, and 40GB of disk.

~~As mentioned, the~~ The test configurations using Ranflood are 72. In addition to these, we gather baseline rate-loss values for each ransomware, run without Ranflood, totalling 78 configurations. We run each configuration 4 times for a total of 312 runs and gather the results for each scenario as the average of the related runs.

Each test run follows the steps:

1. we start the virtual machine and wait that the environment is ready to run the set malware of the run and Ranflood (i.e., we wait for Windows to boot properly);
2. we start the ransomware sample and wait for the set delay of the run;
3. we start Ranflood (Windows native version) with the set flooding strategy of the run. To maximise resource occupation, we launch all 13 flooding instances in parallel, each targeting the sensible folders mentioned in Section 5.1;
4. after 10 minutes since we started the virtual machine, we shut it down;
5. we access the disk from the gateway and run an analyser, called the Filechecker (available as a companion, open-source tool to Ranflood[3]) to calculate the data points of the run;
6. we delete the virtual machine and start the next test run.

Notably, the Filechecker can restore the system to the state before the attack by removing all files except the original, valid ones and the decoy ones, that it can use to replace the originals, if lost (this requires the usage of some copy-based flooding strategy, cf. Section 3.2.4). Concretely, the Filechecker includes two phases. First, before an attack, it records all the signatures (hashes) of the files in the target directories in a reference database (this is similar to how OTF snapshooting works, cf. Algorithm 2). Second, after an attack, it checks the files present on the disk against the recorded signatures. The Filechecker preserves a file if its signature corresponds to a recorded one. In the case of decoy files that are copies of the original ones (which have a different path than the one corresponding to a recorded signature), if the original is missing we replace it with the copy.

5.3. Results and Analysis

The complete set of data gathered from our experiments is available at `https://doi.org/10.5281/zenodo.6587519`. We report the results of our tests in Figure 5, as percentages of lost, saved, and copied files in each attack scenario. For the sake of clarity, we included only the average result computed across the multiple runs of each test, because the standard deviation is ~~very low in almost all~~ generally low among the cases. Specifically, the highest standard deviation occurs in tests related to Phobos, whose average percentage standard deviation is ca. 8% (with a standard deviation of that average of 13).

The cells in Figure 5 are composed as follows: the central area shows the percentage of valid (non-encrypted) files. Since copy-based flooding strategies allow the restoration of lost original files, we break down the percentage of valid files into a blue one (original) and green one (restored), reporting the related percentages respectively at the bottom and at the top of the bar. The red part completes the picture, representing the percentage lost.

The first pieces of ransomware we comment on are GandCrab (GC), Ryuk, and Vipasana, which share ~~a very~~ similar behaviour and thus can be reported as one, for the sake of brevity. They encrypt only files that we do not consider as being sensitive for the user (i.e., outside ~~of~~ the 13 folders monitored by the test cf. Section 5.1). Hence, ~~the report is~~

we report 100% saved files.

LockBit encrypts files following a ~~different strategy , in which~~ strategy where the malware quickly skims through the folders of the user, only encrypting the first 4 KB of each file. This behaviour, in unison with the relatively slow response of Ranflood (which in our tests is set to start, at the earliest, 30 seconds after the activation of the ransomware) makes LockBit the toughest ~~opponent against Ranflood—in the~~ among the opponents—in the future we intend to deepen our research on this kind of attack modality, e.g., proposing ad-hoc, copy-based strategies able to quickly contrast the malware by restoring just the compromised portion of the encrypted files. Both the Random and On-The-Fly strategies fail to contrast it—the ransomware leaves a constant 9% of valid files, which it does not consider as its targets (e.g., configuration files). The Shadow strategy is the only one able to partially hinder LockBit (reaching a 48% of recovery of only copied files) since it uses separate copies of the files for the flooding.

Phobos is designed to encrypt all files in the system when no countermeasure is put in place, as shown by the 0% of valid files in the "None" column of the figure. The interaction with the Random strategy shows an unexpected pattern. ~~The earliest activation of Ranflood~~ Its earliest activation achieves the lowest score (0%), while late activations produce better results, yet not ~~according~~ amounting to some regular pattern: the percentage of valid files jumps to 13% when the delay is 60 seconds, decreases to 10% for 180 seconds, to reach the best value of 14% for 300 seconds. We attribute this behaviour to some internal delays of the ransomware (e.g., to elude detection), which makes the 60s and 300s activation time the fittest to contrast it. This phenomenon is ~~somewhat~~ more or less repeated in the Shadow modality, where the 30-second delay achieves a 22% recovery while the later 60-second delay reaches 29%, before falling to a meagre 2% for higher delays.

WannaCry behaves like LockBit, but it is less aggressive, leaving more than half of the user's files untouched when left free to roam (see ~~as usual~~ the "None" column). ~~Ranflood obtains its highest effectiveness against WannaCry~~ Among our ransomware samples, WannaCry seems the one which Ranflood can contrast the best. Similarly to Phobos, we notice that the we hit the "sweet spot" for the activation delay ~~is probably hit when it collides~~ when it matches with some internal ~~parameter~~ delay of the malware. The effectiveness of the Random modality peaks at 73% saved files when activated with a 180-second delay, On-The-Fly peaks at 67% saved files when activated with a 60-second delay, and Shadow reaches 94% ~~when its activation is at the earliest considered~~ at its earliest activation time.

Copy-based Overhead and Restoration— Aside from the performance benchmarks of the mitigation, we benchmark both the initial overhead derived from the snapshooting routines of the On-The-Fly and Shadow flooding strategies and the performance of the Filechecker (i.e., ~~of the~~ a possible implementation of the restoration phase). In particular, the former is interesting to describe the footprint of the software during the normal operations of the user.

	Avg. (s)	SD (s)
OTF snapshooting	22.15	13.96
Shadow snapshooting	38.69	12.23
Filechecker restoration	573.9	18.38

Table 2
Average time and standard deviation in seconds of the copy-based ~~snapshotting~~ snapshooting and restoration (Filechecker).

We present the performances in Table 2 as the average over eight experiments and the standard deviation of these samples (we report the baseline in the first row (30 sec.) of each table for reference).

~~In particular, we~~ We deem the overhead of both the On-The-Fly and the Shadow strategies compatible with the regular operations of users (interactive) and servers (batch), as they allow for other processes to execute concurrently and do not take a lot of time to complete—this is not different from having an antivirus scan running alongside other processes.

Finally, we notice that the reported measures have a small-yet-non-negligible standard deviation. Indeed, the measures are influenced by ~~a number of~~ several factors which increases the stability of the performance. In particular, regarding the ~~stability of~~ performance of the Filechecker, we notice:

- differences between the operating systems: the Filechecker runs ~~Linux mounting~~ on Linux, where we mount the NTFS disk of the virtual machine through the "qcow2" driver, while the signatures and archive generations run directly in the Windows virtual machine~~passing through~~ , using the virtual device;
- scheduling and parallelism: the Filechecker runs in sequential mode while the signatures and archive generation run in a multithreading application.

~~We argue that investigating the impact of these factors and increasing the performance of the Filechecker elude the scope of this paper and will be subject for future work with a more in-depth study to analyse the performance differences between the drivers and virtualized operating systems.~~ While these performance results are encouraging, we deem an important future work setting out specific tests that would allow us to profile the algorithms and runtimes of the tools, refine them, and increase their performance.

5.4. Comparison with Empirical Evaluations of Related Work

To conclude our empirical assessment of Ranflood, we put our results in perspective against those from empirical evaluations of related work. In doing so, we underline that it is not possible to directly compare the results of the considered evaluations, given that they have been drawn from diverse hardware and software settings, on different sets of ransomware samples, and with disparate experimental set-ups. Moreover, the considered tools are sensibly different in terms of the

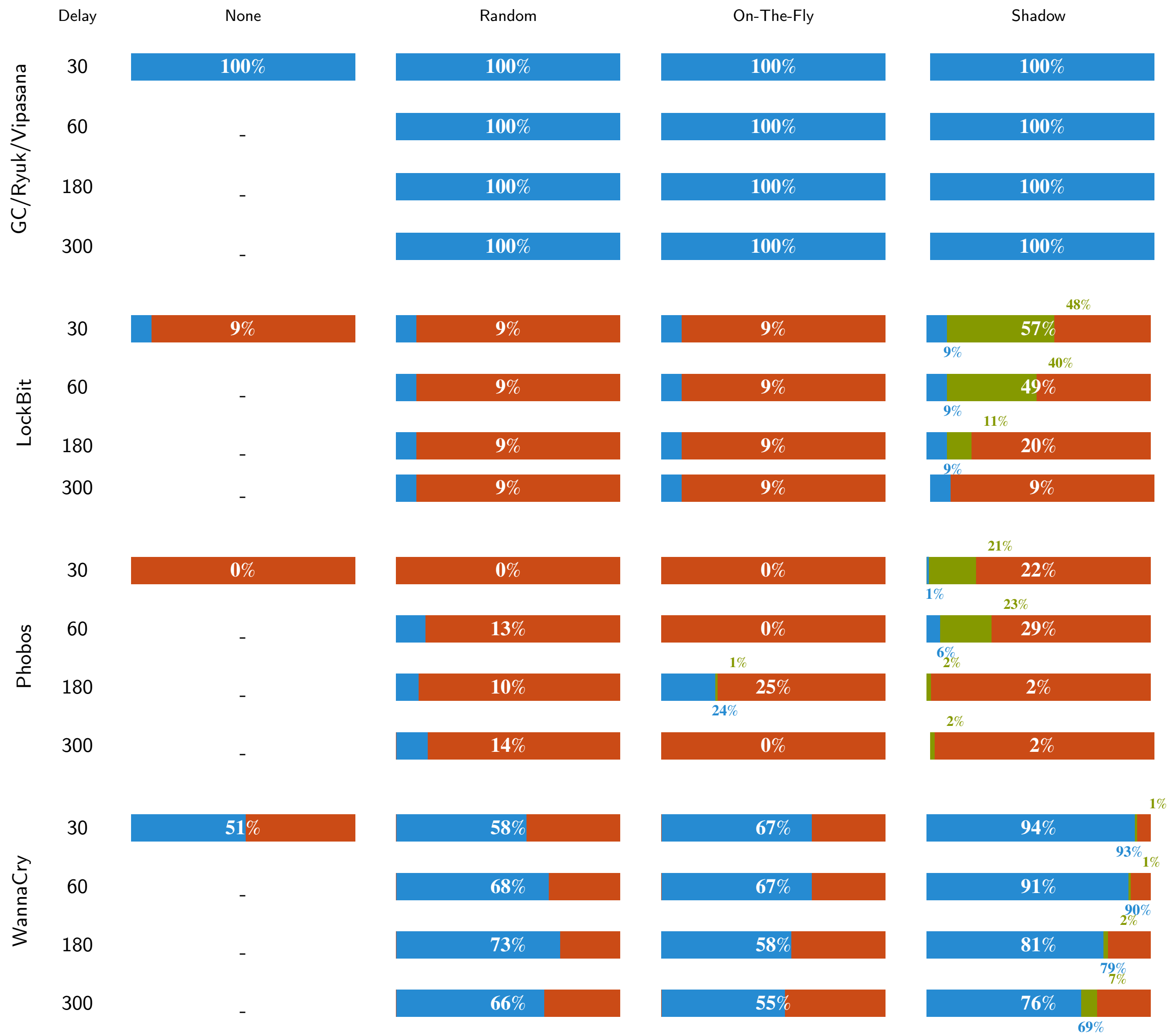

Figure 5: Results of the aggregated tests, loss-rate percentage—each cell shows the percentage of valid (non-encrypted) files. For copy-based strategies we break down the percentage of valid files into a blue one (original) and a green one (restored), reporting the related percentages respectively at the bottom and at the top of the bar. The longer the blue/green bar, the better.

phases they target to contrast ransomware (detection, mitigation, restoration), the technique they rely upon, and the usage requirements—e.g., a solution like Ranflood is closer to installing an antivirus while e.g., ShieldFS is a more involved one, which requires the user to recompile the operating system kernel.

Considering the works covered in Section 2, summarised in Table 1, we compare with those proposals that, like Ranflood, are marked as generic (not tailored to any specific ransomware family) and that implement the mitigation and/or restoration phases. These requirements give us four items: ShieldFS (Continella et al., 2016), R-Locker (Gómez-Hernández et al., 2018), the tool by Lee at al. (Lee et al., 2019), and Microsoft controlled folder access (Microsoft, 2022). Unfortunately, we could not retrieve experimental data regarding the last item (Microsoft's), excluding it from this comparison.

ShieldFS The evaluation done by Continella et al. (2016) comes the closest to ours, since they also measure the performance based on the ratio of recovered data. Thanks to its detection and shadowing capabilities, ShieldFS reaches an aggregated recovery rate of more than 90% (the authors do not provide the breakdown of the considered ransomware families). Quantitatively, aggregating the data from our experiments gives us an 80% recovery rate for Ranflood. Notwithstanding the good figures of the two proposals, we stress that our comparison can only be at the qualitative level, because quantitative comparisons would entail the definition of common testing environments and infrastructures.

R-Locker R-Locker implements a detection and mitigation mechanism, based on the distribution/spread of honeypot files used for both the detection and mitigation phases. The authors only report the aggregated detection rate, 100%, but do not report the ratio of saved-vs-lost files. While the reported figure is impressive, there is a caveat, reported by the same authors, which is that the detection phase can be bypassed by any ransomware that encrypts the files randomly, making the performance drop significantly. Since Gómez-Hernández et al. (2018) focus on the performance of detection while we benchmark the mitigation phase, we cannot directly compare with their results.

Tool by Lee et al. The tool by Lee et al. implements a Moving Target Defence strategy, based on changing the type or extension of the file to deceive the ransomware. Lee et al. report aggregated data as “defence rate”, where they preemptively run their solution (changing the type and extension of a set of selected files), then, they let the ransomware run for 5 minutes and calculate the number of encrypted files. They report a total of 98.6% “defence rate”.

Also comparing our evalutation of Ranflood and that of Lee et al. is difficult, since the latter run the tool before the ransomware, while we test Ranflood after the ransomware started the attack, simulating the triggering from a detector.

6. Discussion and Conclusion

We presented Data Flooding against Ransomware (DFaR) as a family of methods to contrast ransomware that mixes dynamic honeypots, resource contention, and moving target defence. We detailed the three phases of detection, mitigation, and restoration of DFaR. To show the applicability of DFaR we also introduced instantiations of the mitigation and restoration phases as implemented within a tool called Ranflood—specifically Ranflood implements three flooding strategies of which two enable the restoration phase. We also showed preliminary but thorough benchmarks that demonstrate that Ranflood (and its three flooding strategies) is effective in contrasting the action of different kinds of ransomware.

Ranflood is more of a stepping stone than the end of the road. Indeed, as presented in Section 3.2.2, one can use DFaR to detect ransomware. Future work in this direction goes towards studying different instantiations of the DFaR detection paradigm and investigating their interplay: *a*) developing work similar to the one we undertook with Ranflood—implementing and empirically studying the effectiveness of the static and dynamic modalities of detection (cf. Section 3.2.2); *b*) investigating ways of mixing DFaR detection with other existing approaches from the literature, in particular, to implement the patrolling process of the dynamic modality; *c*) testing the effectiveness of detection instantiations based on different combinations of the dynamic and static modalities, depending on disparate platforms of execution, contexts of application, and ransomware families.

Exfiltration ransomware While, in this work, we focussed on crypto-ransomware, there is another growing category of ransomware that is becoming more and more threatening for organisation: exfiltration-based ransomware. Indeed, given the constant threat by crypto-ransomware, organisations started contrasting them with ~~reliable backup systems, which~~ backup plans. Of course, the latter do not hinder ~~their diffusion , but~~ the diffusion of ransomware, but they curb the motivation ~~for~~ of the attackers to strike~~: the victims do not pay any more, since~~ ; the victims are less likely to pay if they can restore (most of) their encrypted files from backups. This motivated the recent surge of new *exfiltration* ransomware, whose objective is not to prevent users from accessing their data but to abduct their sensitive files and threaten to disclose their contents, unless the victims pay the proverbial ransom (Michael, 2021). While currently tailored for crypto-ransomware, we conjecture that ~~the Random strategy (cf. Section 4)~~ DFaR and Ranflood can also effectively contrast exfiltration-based attacks ~~, since it can induce~~ by inducing the malware to transmit decoy files rather than those of the user. In the process, ~~it wastes~~ the tool would make the ransomware waste disk and network IO access, slowing down the exfiltration of worthy payload. Given the rising importance of exfiltration-based attacks, we envision future work also in this direction. ~~The main work~~Work, here, ~~regards the introduction and benchmarking of~~ can start by benchmarking the performance of ~~(i. e., the effectiveness in preventing data exfiltration) a set of Random flooding strategies , which employ different algorithms for the synthesis of decoy files and logics for structuring folders and allocating files.~~ the available flooding strategies of Ranflood in limiting data exfiltration. Then, one can introduce new or refined version of the presented flooding strategies to maximise the contrast they provide against exfiltration-based attacks (e.g., on the content of decoy files, their folders layouts, etc.). To do this, advanced versions of Ranflood (in synergy with detectors) can profile the type of malware that is attacking and tune flooding strategies that minimise its effect. For example, one can refrain from using copy-based strategies when dealing with exfiltration, to avoid the possibility of providing sensible content to the ransomware via decoy copies of the actual files of the user. However, we underline that the matter can more nuanced than this. Indeed, when we induce the ransomware to exfiltrate the same content over and over, we are making the ransomware waste time and bandwidth to obtain the same information. Future work on exfiltration ransomware shall investigate this matter, e.g., quantify the ratio between exfiltrated content and wasted bandwidth/time due to copy-based flooding strategies.

On a more general note, we foresee studying the interplay between detection and mitigation, so that the former can tune the flooding strategy of the latter. The main example, here, is a detector that “understands” the patterns of the attacking ransomware, and informs the mitigation to use specific flooding modalities that have been empirically demonstrated to work best against that kind of ransomware. Referring to the previous paragraph, a detector able to discriminate between crypto- and exfiltration-based ransomware can instruct the mitigation tool to use copy-based strategies rather than random-based ones.

Besides investigating the functional aspects of DFaR solutions, we deem it important to study the aspects related to human-computer interaction with Ranflood and other DFaR-based prototypes. These aspects include letting the user know when a detection instance starts, on which folders the detector operates, and what files the software creates as decoys. The same goes for the mitigation, where we should inform the user of the ongoing attack and the fact that the software is flooding which folders of the attacked machine. Experiments should investigate both what are the best techniques to communicate this information to the user and what are the best ways to stimulate the user in adopting secure behaviour, e.g., to inform users of the ongoing attack and report the issue to system administrators.

Finally, future work can focus on the restoration phase of DFaR, e.g., following the idea of implementing a fingerprinting feature in the mitigation and restoration ~~phase~~phases, which dispenses the user from relying on additional resources than the decoy files themselves (cf. Section 3.2.4). ~~For instance, if the resource is lost, the user cannot perform the restoration step; as an example, this case was represented by the list of file signatures we depended upon for the execution of the restoration tool we benchmarked in this paper (losing that file would~~ This is exemplified by our naïve implementations—e.g., the On-The-Fly copy-based strategy and the restoration tool (Filechecker)—which rely on a list of signatures of the original files, whose loss could prevent us from ~~carry out with the restoration phase)~~executing the flooding/restoration step in our experiments.

References

Lawrence Abrams. 2021. Ransomware attack hits Italy's Lazio region, affects COVID-19 site. https://www.bleepingcomputer.com/news/security/ransomware-attack-hits-italys-lazio-region-affects-covid-19-site/

Yahye Abukar Ahmed, Baris Kocer, and Bander Ali Saleh Al-rimy. 2020. Automated analysis approach for the detection of high survivable ransomware. *KSII Transactions on Internet and Information Systems (TIIS)* 14, 5 (2020), 2236–2257.

Maxat Akbanov, Vassilios G. Vassilakis, and Michael D. Logothetis. 2019. Ransomware detection and mitigation using software-defined networking: The case of WannaCry. *Computers & Electrical Engineering* 76 (2019), 111–121. https://doi.org/10.1016/j.compeleceng.2019.03.012

Bander Ali Saleh Al-rimy, Mohd Aizaini Maarof, and Syed Zainudeen Mohd Shaid. 2018. Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions. *Computers & Security* 74 (2018), 144–166.

Blake Anderson and David McGrew. 2016. Identifying Encrypted Malware Traffic with Contextual Flow Data. In *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security* (Vienna, Austria) *(AISec '16)*. Association for Computing Machinery, New York, NY, USA, 35–46. https://doi.org/10.1145/2996758.2996768

Nicoló Andronio, Stefano Zanero, and Federico Maggi. 2015. HelDroid: Dissecting and Detecting Mobile Ransomware. In *Research in Attacks, Intrusions, and Defenses*, Herbert Bos, Fabian Monrose, and Gregory Blanc (Eds.). Springer International Publishing, Cham, 382–404.

Krzysztof Cabaj, Marcin Gregorczyk, and Wojciech Mazurczyk. 2018. Software-defined networking-based crypto ransomware detection using HTTP traffic characteristics. *Computers & Electrical Engineering* 66 (2018), 353–368. https://doi.org/10.1016/j.compeleceng.2017.10.012

Bill Chappell and Colin Dwyer. 2017. Massive ransomware attack hits Ukraine; experts say it's spreading globally. https://www.npr.org/sections/thetwo-way/2017/06/27/534560169/large-cyberattack-hits-ukraine-snarling-electric-grids-and-airports?t=1643028558133

Christopher JW Chew and Vimal Kumar. 2019. Behaviour based ransomware detection. (2019).

Andrea Continella, Alessandro Guagnelli, Giovanni Zingaro, Giulio De Pasquale, Alessandro Barenghi, Stefano Zanero, and Federico Maggi. 2016. ShieldFS: a self-healing, ransomware-aware filesystem. In *Proceedings of the 32nd Annual Conference on Computer Security Applications, ACSAC 2016, Los Angeles, CA, USA, December 5-9, 2016*, Stephen Schwab, William K. Robertson, and Davide Balzarotti (Eds.). ACM, 336–347. http://dl.acm.org/citation.cfm?id=2991110

Ahmed El-Kosairy and Marianne A Azer. 2018. Intrusion and ransomware detection system. In *2018 1st International Conference on Computer Applications & Information Security (ICCAIS)*. IEEE, 1–7.

David Evans, Anh Nguyen-Tuong, and John Knight. 2011. Effectiveness of moving target defenses. In *Moving target defense*. Springer, 29–48.

Amirhossein Gharib and Ali Ghorbani. 2017. Dna-droid: A real-time android ransomware detection framework. In *International Conference on Network and System Security*. Springer, 184–198.

José Antonio Gómez-Hernández, L Álvarez-González, and Pedro García-Teodoro. 2018. R-Locker: Thwarting ransomware action through a honeyfile-based approach. *Computers & Security* 73 (2018), 389–398.

Samuel Greengard. 2021. The worsening state of Ransomware. https://cacm.acm.org/news/251337-the-worsening-state-of-ransomware/fulltext

Andrew Griffin. 2017. 'Petya' cyber attack: Chernobyl's radiation monitoring system hit by worldwide hack. https://techbeacon.com/security/ransomware-rise-evolution-cyberattack.

Mike Halsey. 2016. *Windows 10 File Structure in Depth*. Apress, Berkeley, CA, 449–457. https://doi.org/10.1007/978-1-4842-0925-7_27

Nihad A Hassan. 2019. Ransomware Decryption Tools. In *Ransomware Revealed*. Springer, 191–201.

Casen Hunger, Mikhail Kazdagli, Ankit Rawat, Alex Dimakis, Sriram Vishwanath, and Mohit Tiwari. 2015. Understanding contention-based channels and using them for defense. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 639–650.

Carroll Joe, Guerra Luz Andres, and R Shah Jill. 2021. Gas Stations Run Dry as Pipeline Races to Recover From Hacking - Bloomberg. https://www.bloomberg.com/news/articles/2021-05-09/u-s-fuel-sellers-scramble-for-alternatives-to-hacked-pipeline.

Kaspersky. 2021. *Ransomware Attacks and Types*. https://www.kaspersky.com/resource-center/threats/ransomware-attacks-and-types

Amin Kharaz, Sajjad Arshad, Collin Mulliner, William Robertson, and Engin Kirda. 2016. UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 757–772. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/kharaz

Amin Kharraz, William Robertson, Davide Balzarotti, Leyla Bilge, and Engin Kirda. 2015. Cutting the Gordian Knot: A Look Under the Hood of Ransomware Attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, Magnus Almgren, Vincenzo Gulisano, and Federico Maggi (Eds.). Springer International Publishing, Cham, 3–24.

S Kok, Azween Abdullah, N Jhanjhi, and Mahadevan Supramaniam. 2019. Ransomware, threat and detection techniques: A review. *International Journal of Computer Science and Network Security* 19, 2 (2019), 136.

Eugene Kolodenker, William Koch, Gianluca Stringhini, and Manuel Egele. 2017. Paybreak: Defense against cryptographic ransomware. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. 599–611.

Kroll. 2021. *Data Exfiltration Ransomware Attacks*. https://www.kroll.com/en/insights/publications/cyber/data-exfiltration-ransomware-attacks

Kyungroul Lee, Kangbin Yim, and Jung Taek Seo. 2018. Ransomware prevention technique using key backup. *Concurrency and Computation: Practice and Experience* 30, 3 (2018), e4337.

Suhyeon Lee, Huy Kang Kim, and Kyounggon Kim. 2019. Ransomware protection using the moving target defense perspective. *Comput. Electr. Eng.* 78 (2019), 288–299. https://doi.org/10.1016/j.compeleceng.2019.07.014

Allan Liska and Timothy Gallo. 2016. *Ransomware: Defending against digital extortion*. " O'Reilly Media, Inc.".

George Marsaglia. 2003. Xorshift RNGs. *Journal of Statistical Software* 8, 14 (2003), 1–6. https://doi.org/10.18637/jss.v008.i14

Shagufta Mehnaz, Anand Mudgerikar, and Elisa Bertino. 2018. Rwguard: A real-time detection system against cryptographic ransomware. In *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 114–136.

Melissa Michael. 2021. Episode 49: Ransomware 2.0, with Mikko Hypponen - F-Secure blog. https://blog.f-secure.com/podcast-ransomware-mikko/

Microsoft. 2022. Protect important folders with controlled folder access. https://docs.microsoft.com/en-us/microsoft-365/security/defender-endpoint/controlled-folders?view=o365-worldwide.

Chris Moore. 2016. Detecting ransomware with honeypot techniques. In *2016 Cybersecurity and Cyberforensics Conference (CCC)*. IEEE, 77–81.

Routa Moussaileb, Benjamin Bouget, Aurélien Palisse, Hélène Le Bouder, Nora Cuppens, and Jean-Louis Lanet. 2018. Ransomware's early mitigation mechanisms. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*. 1–10.

Routa Moussaileb, Nora Cuppens, Jean-Louis Lanet, and Hélène Le Bouder. 2021. A survey on windows-based ransomware taxonomy and detection mechanisms. *ACM Computing Surveys (CSUR)* 54, 6 (2021), 1–36.

Nicole Perlroth, Mark Scott, and Sheera Frenkel. 2017. Cyberattack hits Ukraine then spreads internationally. https://www.nytimes.com/2017/06/27/technology/ransomware-hackers.html

Person and Conor Humphries Padraic Halpin. 2021. Irish Health Service hit by 'very sophisticated' ransomware attack. https://www.reuters.com/technology/irish-health-service-hit-by-ransomware-attack-vaccine-rollout-unaffected-2021-05-14/

Irfan Pyarali, Tim Harrison, Douglas C. Schmidt, and Thomas D. Jordan. 1997. *Proactor - An Object Behavioral Pattern for Demultiplexing and Dispatching Handlers for Asynchronous Events*. Technical Report. Washington University.

Christian Rossow, Christian J Dietrich, Chris Grier, Christian Kreibich, Vern Paxson, Norbert Pohlmann, Herbert Bos, and Maarten Van Steen. 2012. Prudent practices for designing malware experiments: Status quo and outlook. In *2012 IEEE symposium on security and privacy*. IEEE, 65–79.

Nolen Scaife, Henry Carter, Patrick Traynor, and Kevin R. B. Butler. 2016a. CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data. In *36th IEEE International Conference on Distributed Computing Systems, ICDCS 2016, Nara, Japan, June 27-30, 2016*. IEEE Computer Society, 303–312. https://doi.org/10.1109/ICDCS.2016.46

Nolen Scaife, Henry Carter, Patrick Traynor, and Kevin R. B. Butler. 2016b. CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. 303–312. https://doi.org/10.1109/ICDCS.2016.46

Millar Sheila A. and Marshall Tracy P. 2017. WannaCry: Are your security tools up to date? https://www.natlawreview.com/article/wannacry-are-your-security-tools-to-date

Douglas Robert Stinson and Maura Paterson. 2018. *Cryptography: Theory and Practice (Textbooks in Mathematics)* (hardcover ed.). Chapman and Hall/CRC. 598 pages.

Andrew Tanenbaum. 2009. *Modern operating systems*. Pearson Education, Inc.,.

Lena Y. Connolly and David S. Wall. 2019. The rise of crypto-ransomware in a changing cybercrime landscape: Taxonomising countermeasures. *Computers & Security* 87 (2019), 101568. https://doi.org/10.1016/j.cose.2019.101568

Hiba Zuhair and Ali Selamat. 2019. RANDS: A machine learning-based anti-ransomware tool for windows platforms. In *Advancing Technology Industrialization Through Intelligent Software Methodologies, Tools and Techniques*. IOS Press, 573–587.