Ranflood: A mitigation tool based on the principles of data flooding against ransomware

Davide Berardi, Saverio Giallorenzo, Andrea Melis, Simone Melloni, Marco Prandini

SoftwareX 25 (2024) 101605

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

Ranflood: A mitigation tool based on the principles of data flooding against ransomware

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

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

ARTICLE INFO

Keywords:
Ransomware
Dynamic honeypot
Moving Target Defence
Data Flooding Against Ransomware

ABSTRACT

Crypto-ransomware aims at extorting money from users by encrypting their files and asking them to pay for the decryption key. We present *Ranflood*; a configurable drop-in solution that contrasts ransomware attacks with a deluge of decoy files at specific locations (e.g., sensitive folders of the user, the attack site), deceiving the attacker into encrypting sacrificial files. Ranflood further slows down the attack by contending with the malware access to IO and computation resources of the targeted machine. The aim is to buy time for the defence team to take action (e.g., manually shutting down an unresponsive machine). We show how the extensibility and modularity of Ranflood's software architecture (1) can accommodate a wide spectrum of flooding strategies, easing the process of improving its effectiveness also against future ransomware families and (2) strive to maximise the tool's efficiency by exploiting the highest level of parallelism afforded by the attacked machine.

Code metadata

Current code version	0.5.9-beta
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-23-00396
Legal Code License	GNU Lesser General Public License v2.1
Code versioning system used	git
Software code languages, tools, and services used	Java 17, reactivex.rxjava3, jetbrains.xodus, apache.commons-io, apache.commons-compress
Compilation requirements, operating environments & dependencies	Compilers: JDK 17, GraalVM Native Image 21. Operating Systems: Linux, macOs, Windows.
Permanent link to executables of this version	https://github.com/thesave/ranflood/releases/tag/0.5.9-beta
Legal Software License	GNU Lesser General Public License v2.1
Computing platforms/Operating Systems	Linux, macOs, Windows
Installation requirements & dependencies	JVM (in non-natively supported operating systems)

1. Motivation and significance

Ransomware is a category of malicious software that extorts resources (usually, money) from users to regain control over their digital property [1]. Ransomware does not come in one form and attackers have multiple ways to take users' resources hostage. The most rampant and popular way (so much so that it frequently makes it to general-audience headlines) is *crypto-ransomware*, where the resources under attack are the data of the user and the method to expropriate them is encryption [2]. Once an attacker has encrypted a victim's data, they ask for a ransom to provide the key that the victim can use to decrypt and regain access to their data. Over the past five years, there has been a significant increase in the frequency of attacks, many of them with severe repercussions on people and businesses, such as the *NotPetya* attack in 2017, [3], the US Colonial Pipeline case in 2021 [4], and various episodes involving healthcare [5–7].

* Corresponding author at: Alma Mater Studiorum – Università di Bologna, Italy.
E-mail address: saverio.giallorenzo@gmail.com (Saverio Giallorenzo).

https://doi.org/10.1016/j.softx.2023.101605
Received 27 June 2023; Received in revised form 29 November 2023; Accepted 3 December 2023
Available online 22 December 2023

In this work, we present Ranflood, a software tool to contrast ransomware attacks, particularly tailored for the crypto-ransomware kind. Ranflood implements the concepts of Data Flooding Against Ransomware [8] (DFaR). DFaR evolves from honeypots, a technique of software security where sacrificial resources are deliberately made vulnerable to attacks in order to detect and deter intruders. Ranflood contrasts attacking ransomware by flooding specific locations of the disk (e.g., the attack location, user folders, etc.) with honeypot decoy files. Essentially, Ranflood buys time for the user to counteract an ongoing attack, e.g., to access an unresponsive, attacked server and shut it down manually. In detail, Ranflood implements a *dynamic honeypot* approach, which consists in generating decoy files at runtime-defined locations 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). In particular, Ranflood is a last-resort solution, useful when more direct actions, like shutting down the malicious process or the whole machine, are unfeasible. Summarising, Ranflood develops three main traits to mitigate ransomware.

Resource Contention Ransomware contrast happens by contending resources (access to IO, memory usage). The high rate of data flooding exerted by Ranflood makes it a contender for both CPU, memory, and disk access against ransomware.

Moving Target Defence By confounding the data of the user with decoy files, Ranflood performs a moving-target defence, which makes it harder for ransomware to efficiently perform their attack on the user's data since they struggle to discriminate between the former and the decoys.

Dynamic Honeypot Since Ranflood's contrast locations are determined at runtime, its defence surface is dynamic and can adapt to the attack profiles of different ransomware implementations.

In Section 2, we describe the relevant characteristics of Ranflood, namely, its three flooding strategies, its software architecture, and its functionalities. In Section 3, we illustrate the usage of Ranflood by means of an example. We show different configuration profiles to contrast the famous WannaCry ransomware. We present benchmarks of the effectiveness of the tool and the efficiency of the different configurations. In Sections 4 and 5, we respectively comment on the impact of Ranflood and summarise its current state and discuss its evolution. In the spirit of this publication venue, this article focuses on the practical implementation of the Ranflood tool. We refer the reader interested in having more details about the concepts and related work, which are only briefly summarised here, to the article dedicated to introducing DFaR [8].

Related work. The literature on ransomware contrast is vast. We can broadly categorise work as addressing the detection, mitigation, and restoration phases. Many related works focus on the detection phase and recovery from known widely analysed ransomware. Contrarily, Ranflood is a general, drop-in solution that focuses on the mitigation phase of a ransomware attack. Since Ranflood implements a mitigation and restoration technique against ransomware, for space reasons, we briefly compare it with alternative techniques from the literature that implement those phases. Here, we select proposals that, like Ranflood, are generic (not tailored to any specific ransomware family) and that implement the mitigation and/or restoration phases: ShieldFS [9], R-Locker [10], the tool by Lee at al. [11], and Microsoft's Controlled Folder Access [12].

ShieldFS relies on the integration between an ad-hoc file system and a detector (not integral to ShieldFS, which can work with different detectors). When the detector recognises a ransomware attack, 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. Ranflood can also support restoration via its copy-based strategies (cf. Section 2.1). Contrarily to ShieldFS, Ranflood requires little preliminary configuration (similar to mainstream drop-in security tools, like antiviruses); far less involved than installing an ad-hoc file system.

R-Locker implements a detection and mitigation mechanism, based on the distribution/spread of honeypot files used for both the detection and mitigation phases. At the moment, R-Locker is designed for the Linux system, while Ranflood is natively compiled for Windows, macOS, and Linux, and can run on any architecture with support for the Java Virtual Machine.

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 [13]. Also the flooding action performed by Ranflood follows a Moving Target Defence strategy, with the addition of resource contention, which further mitigates the action of the malware.

The principle exploited by Microsoft's solution is that 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.

We report the key points of the comparison in Table 1.

2. Software description

We present and discuss Ranflood's flooding strategies (Section 2.1), software architecture (Section 2.2), functionalities (Section 2.3), and current limitations (Section 2.4).

2.1. Three flooding strategies

We start by briefly overviewing the three flooding strategies the Ranflood engine realises, breaking them down into the operations and traits that characterise them. Broadly, we distinguish between two kinds of flooding strategies. The first is configuration-less and does not require any preliminary action to trigger the flooding, represented by the Random strategy (cf. Section 2.1.1). The second uses copies of the genuine files of the user to decrease the loss rate of files due to encryption, but it entails preliminary actions like storing the user's file signatures or making archival snapshot copies of the latter—On-The-Fly (cf. Section 2.1.2) and Shadow (cf. Section 2.1.3) conform to this kind of strategy.

2.1.1. Random

The first strategy is called *Random* and it is based on the flooding of a given location with randomly-generated files.

While one can propose more refined strategies (like the one we discuss below) which can increase the efficiency of the contrast action of Ranflood, the Random one has the relevant characteristic of being easy to deploy and use. Indeed, this strategy includes only one operation, i.e., the flooding one, and it is also configuration-less (e.g., it does not require the user to provide dedicated locations for the auxiliary data needed by copy-based alternatives).

The Random strategy has the following three main peculiarities. First, each generated file's content is unique, making it difficult for ransomware that tries to avoid encrypting decoy files to distinguish between the proper files of the user and the generated ones. Second, it produces large amounts of decoy files in a short timeframe. Third, it generates files using extensions that ransomware usually target [9,14,15] (e.g., it produces files formatted as and with extensions of common formats such as ".pdf" and ".jpg").

The current implementation of the Random strategy achieves the above characteristics by: using a variant of Xorshift [16] to obtain many random byte sequences quickly (their lengths are random too, within an interval that the user can configure); matching the format of the file (declared by its extension) and its binary header; generating files at random file paths (folder and file name).

Table 1
Table comparing related work. Each row in the table corresponds to a strategy found in works related to ours—the last row corresponds to this article, for comparison—reported in the rightmost column. The other columns show properties of the principle behind each tool: to what phases it applies (detection, mitigation, restoration) 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).

Principle	Detection	Mitigation	Restoration	Drop-in solution	References
Restrict permissions	○	●	○	●	[12]
Extension randomisation	○	●	○	○	[11]
Honeypot files	●	●	○	○	[10]
Self-healing file system	○	○	●	○	[9]
Data flooding	○	●	● [a]	●	**Ranflood**

[a] Copy-based flooding (cf. Section 2.1).

2.1.2. On-The-Fly

The second strategy, called *On-The-Fly*, is copy-based, i.e., we essentially 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.

In the implementation, when copying files, we avoid replicating encrypted ones. Indeed, copying these files is detrimental because it wastes the time of the flooder on files useless to the user, and it generates files that the malware would skip, recognising them as already encrypted.

The above refinement introduces a new operation that precedes the flooding one, which we call "snapshooting". With this operation, the Ranflood engine saves a list of the valid files, later used during flooding for efficient discrimination. In the implementation, we chose to realise the snapshotting operation for this strategy by saving a digest signature (e.g., MD5) of the content of the user files, so that the flooding operations can use the signature as an integrity verification to skip the encrypted files. The implementation of the flooding operation is also a bit more refined than the copy principle we presented above. Indeed, a naïve interpretation is to iterate over all files in the flooding location, read the content of each one and write it in a new, randomly named/located file. However, this logic leaves the possibility of losing files between iterations—e.g., we replicate a user's file f in f' at the first iteration, then the ransomware reaches both files and encrypts them, preventing us from both copying and restoring the content of f. To avert this risk, the engine runs a slightly more sophisticated alternative that caches the content of the files read once from the disk and then iterates their replication (trading memory occupancy off mitigation efficiency).

2.1.3. Shadow

The third strategy, also copy-based, is called *Shadow* and tries to further, increase the efficiency of the On-the-Fly one by preserving an archival copy the files of the user rather than more lightweight information, such as their fingerprint (hence, additionally trading disk occupancy off mitigation efficiency).

Similarly to the On-the-Fly strategy, Shadow has a configuration step and two operations (snapshooting and flooding). In the implementation, we use (tar.gz) archives to try to reduce the space required for snapshots and preserve those archives on the same disk as the original copies, both for simplicity and to shorten loading times. Advanced configurations can use secondary disks, NAS, and the Cloud to further mitigate the possibility of losing the local backups if targeted by ransomware.

2.2. Software architecture

Concerning its architecture, Ranflood follows the well-known client-daemon pattern [17, Chapter 2].

The daemon is an always-active component, and it is used both to take snapshots of files and to execute flooding commands. We expect only one daemon to run on a machine, while several clients can connect to the same daemon. Specifically, the daemon is a process in the background, not associated with a particular user, and users/programs interact with it with lightweight, asynchronous clients/interfaces.

Here, we concentrate on the description of the components of the daemon. We describe the client through the functionalities it provides to users in Section 2.3. The daemon has two main components. One is the *engine*, which realises different flooding strategies and their related operations—for instance, an operation is the flooding of a folder using the Random strategy. Since we want the daemon to be able to handle different operations in parallel, we divide them into tasks, which are minimal actions within the execution flow of a given operation, e.g., an action could be copying a specific file in an On-the-Fly flooding operation. Accordingly, the other main component of the Ranflood daemon is the *task manager*, which handles the scheduling of various tasks. Isolating tasks has the additional benefit of making the implementation of operations more resilient; if a task fails—e.g., because the folder where to generate a file does not exist, resulting in an IO error—it does not affect the operation it belongs to—e.g., the operation continues its ongoing flooding action.

The *engine* implements the operations of the flooding strategies described in Section 2.1, producing their related tasks. The *task manager* handles the tasks as generic work that it schedules for execution. Concretely, we implemented the task manager following the Proactor [18] event-handling pattern. The Proactor decouples the task demultiplexing and the task-handler scheduling logic from the actual behaviour enacted by the single tasks, asynchronously. This execution method helps in further exploiting parallelism and minimising the effect of IO overhead and latency.

We further clarify the architecture of Ranflood by depicting a model of it in Fig. 1. 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

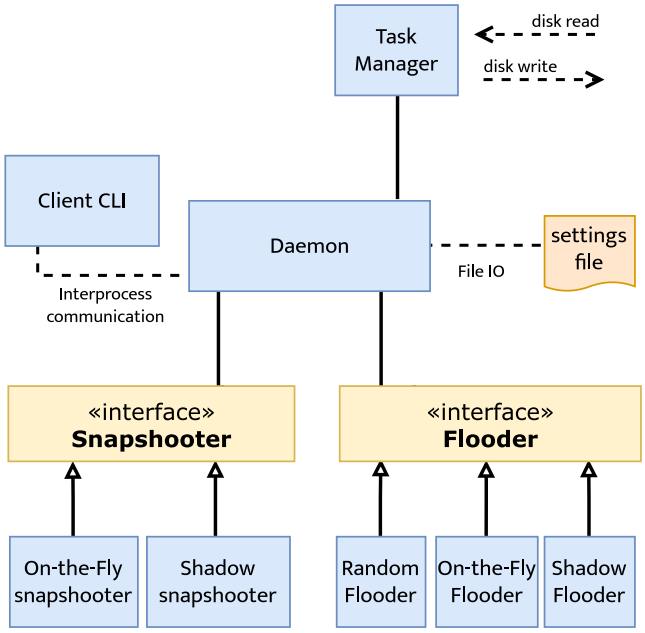

Fig. 1. Model of Ranflood's architecture.

the latter stores in a settings file. The other main components are dedicated to implementing the different flooding strategies, i.e., they realise the Ranflood engine. 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 snapshotting phase, which they realise by implementing the Snapshooter interface. The Daemon interacts with these components to obtain tasks that operate on files, run in parallel by the Task Manager (which implements the Proactor's logic).

Ranflood—both its client and daemon—is an open-source project.[1] written in Java. It uses the RxJava.[2] library for the basic components of its task manager and, through the GraalVM.[3] compiler, it is available as native binaries for Windows, macOS, and Linux systems, besides its Java executable. This article refers to version 0.5.9-beta of Ranflood. 'Code metadata' provides further project metadata.

2.3. Software functionalities

We start discussing the functionalities of Ranflood by overviewing the steps of the main logic implemented by the tool, which is the flooding action against ransomware.

The first step is the triggering of a Ranflood's operation. Here, we choose to adopt an open stance towards the event that can trigger that operation. Indeed, while the user can trigger actions with the CLI, other software can automatise the triggering, e.g., ransomware detection tools, via the supported interprocess communication medium/format. Focusing on flooding, irrespective of the triggering modality, when a flooding operation starts, it floods one or more targeted folders (e.g., where the ransomware is attacking, but also critical locations, independent of where the attack is running, such as the user's folders). This happens until the emission of a signal that stops the flooding (we abstract away the entity triggering this event).

After the mitigation phase above, it is possible to run restoration routines that try to recover the user's environment as before the ransomware attack. This action mainly regards two aspects: (*i*) removing the flooding files and (*ii*) depending on the flooding technique employed, (cf. the copy-based ones found in Section 2.1), restoring the files of the user that might have been encrypted by ransomware.

Let us focus on actions issued by the CLI presented above, assuming the Ranflood daemon is running. Concretely, we can mainly issue commands to the Ranflood daemon with the following three parameters:

- Action: whether we want to `flood` or take a `snapshot` (copy-based strategies);
- Target Folder: the folder where to perform the action;
- Method: the strategy we consider for the action (Random, On-the-Fly, Shadow).

Focussing on the `flood` command, the daemon can perform three sub-actions:

- `ranflood flood list`: list all ongoing flooding operations, identified by a unique id (here, the target folder and method parameters are immaterial);
- `ranflood flood start <method> <targetFolder>`: start a flooding operation (Random, On-the-fly, Shadow) on a specific folder;
- `ranflood flood stop <method> <ids>`: stop a list of flooding operations, identified via their ids (since the id identifies an ongoing operation on a target folder, the latter is irrelevant).

After a ransomware attack, mitigated using copy-based Ranflood strategies, we can run a restoration phase where we try to recover the original environment of the user before the attack, both trying to restore the files of the user lost to encryption and remove the decoy files generated by Ranflood.

Since this phase is distinct from the mitigation one, we implement the needed routines within a separate tool which we only overview, to keep our focus on Ranflood. Future versions of Ranflood could include the additional functionalities of the restoration tool to cover a wider range of the ransomware contrast spectrum. Briefly, the restoration tool compares the files found in the attacked locations with the signatures of the files of the user (obtained before an attack, similar to the snapshotting operations of the On-the-Fly strategy), either finding the original copy of the user or replacing the latter with a flooding-generated replica, if any.

2.4. Limitations on the effectiveness of ranflood against ransomware processes

One way to reduce the effectiveness of Ranflood is if the malware operated at a faster pace than Ranflood. However, we deem this scenario unrealistic. Intuitively, the available Ranflood's strategies perform linear tasks (generating/reading and writing bit-by-bit sequences) and there is no intrinsic computational overhead associated with the selection of files. Note that we implemented Ranflood so that users can combine it with sophisticated routines for determining the flooding locations. These can have different complexity profiles but are orthogonal to the complexity of the Ranflood functionalities, as described above. The only additional overhead is hash computation, performed only once by the On-the-Fly strategy, and its complexity is lower than that of the encryption operation of the ransomware, which is done for every encrypted file.

Another possible way to reduce the effectiveness of Ranflood could result from resource exhaustion, in particular if the disk is full. At disk exhaustion, we conjecture that the most likely case is that the system (including Ranflood and the ransomware) is stuck because no process can write on disk. We deem this case positive since the ransomware cannot do any more harm. However, there could be more refined routines that use bit-by-bit file overwriting which could allow both Ranflood (using the copy-based flooding strategies) and the ransomware to

[1] https://github.com/Flooding-against-Ransomware/ranflood
[2] https://github.com/ReactiveX/RxJava
[3] https://www.graalvm.org/

progress, e.g., by overwriting each other's files. We deem also this case positive since Ranflood would continue to contrast the ransomware and buy time for the user. To the best of our knowledge, there exists no ransomware that employs one such routine, and we did not deem it useful to implement the corresponding logic for Ranflood in its current incarnation.

3. Illustrative example

We complete our overview of Ranflood by showing and commenting on a concrete configuration of the daemon and discussing experimental results we obtained by using it to contrast the infamous WannaCry ransomware.

3.1. Daemon configurations

We report in Listing 1 an example configuration of the Ranflood daemon. The configuration file uses the INI format, where key–value pairs define properties within a given section, marked with square brackets.

```
[RandomFlooder]
MaxFileSize = 768 KB
[OnTheFlyFlooder]
Signature_DB = C:\Users\user_folder
ExcludeFolderNames = Application Data
[ShadowCopyFlooder]
ArchiveDatabase = D:\shadow\archives.db
ArchiveRoot = D:\shadow\archives
ExcludeFolderNames = Application Data
[ZMQ_JSON_Server]
address = tcp://localhost:7890
```

Listing 1: Illustrative configuration file of the Ranflood daemon.

First, notice that each flooding strategy has a dedicated section, where we can adjust features such as generated file size, location of copied files, folder to exclude from the flooding, etc. The idea is to create different settings files for different nodes of a given system. In this way, it is possible to implement different profiles according to the needs of system administrators, the operating system, and the execution environment.

The contents of the configuration file reported in Listing 1 correspond to the setup used in the experiments from [8], whose we report the excerpt regarding WannaCry in the next section.

We briefly describe the properties in Listing 1 and the design decisions behind them.

For the Random strategy, the `MaxFileSize` sets the maximum file size of each generated file. While Ranflood considers the range [8,4096] Kb for randomly determining the size of each generated file, depending on the hardware and operating system, the Mb range might penalise the efficiency of Ranflood, which would produce a smaller number of larger files. We empirically observed this phenomenon in our experiments, which led us to use the 768Kb mark shown in Listing 1 to balance between the quantity and size of the generated files.

For the On-the-Fly strategy, the `SignatureDB` sets the folder where the daemon saves the signatures file snapshots. The key `ExcludeFolderNames` lists the folders that Ranflood has to skip when performing On-the-Fly flooding; in Listing 1 we exclude from the flooding action the "Application Data" folder, which rarely contains user-sensible data but rather program files that one can restore by reinstalling the programs.

For the Shadow Copy strategy, `ArchiveDatabase` sets the location of the database that contains the index of snapshots, and `ArchiveRoot` defines where to store single archival snapshots (`ExcludeFolderNames` follows the same logic as the namesake property of the On-the-Fly strategy).

In the last section, the parameter `address` configures the socket address where we can reach the Ranflood demon.

3.2. Ranflood's contrast effectiveness and efficiency

We conclude this section by reporting empirical results obtained using Ranflood to contrast real-world ransomware.

These results come from a wider and more detailed analysis for evaluating the efficacy and efficiency of Ranflood published in [8]. Those experiments include a set of 6 ransomware samples averaged over 4 runs for each of 72 possible combinations of ransomware, flooding modality, and triggering delay—all the results are available at https://doi.org/10.5281/zenodo.6587519. Before delving into the discussion of the experimental results, we provide the due details of the testbed used to run the experiments. The testbed consists of four desktop computers each equipped with an Intel i3-4170 (3.70 GHz) dual-core, four-threads CPUs, 12 GB of RAM, and a 500 GB HDD. On these machines, we run ProxMox version 7.0–8 on GNU/Linux. The operating system for running the tests is Windows version 10 (x64) Stable 1809 provided by Microsoft. Each node runs one virtual machine with a dual-core, four-thread CPU, 12 GB of RAM, and 40 GB of disk. To run the tests, we set the Ranflood daemon up using the configuration reported in Listing 1, apart from the folders' paths, which we abbreviated in Listing 1 for presentation purposes.

In Fig. 2, we report the performance of the three flooding strategies, given a triggering delay of 30 s after we launched the ransomware WannaCry sample. We fixed an activation delay, which simulates the triggering from a detection routine, to make our tests more consistent. Commenting on the figure, we notice that the Random strategy, while simple, already achieves a 58% recovery rate (i.e., the number of user files that were saved from the attack). The copy-based strategies consistently increase over the latter performance. Specifically, the On-the-Fly strategy reaches a 67% recovery rate while the more expensive (in terms of snapshot disk occupancy) Shadow strategy achieves 94%—these last results include both the mitigation action of the flooding and the effect of the restoration of files through their replicas.

4. Impact

Ranflood is the first open-source implementation of a ransomware contrast tool based on the Data-Flooding-against-Ransomware paradigm. The tool equips three different flooding strategies to contrast ransomware. Ranflood's design makes it highly configurable both regarding how users and other programs can interact with it and for the definition of node-specific settings.

Ranflood concretely proves the feasibility and effectiveness of the DFaR contrast approach, paving the way for more refined tools that could advance the state of the art. Besides these aspects, the modular architecture of Ranflood makes it a workbench to extend the existing flooding strategies, to include new ones, and to develop new functionalities, such as detection and recovery routines.

While we deem the above results satisfactory, Ranflood is an early prototype. Hence, to make it reach a wide audience, we foresee future work focused on performing thorough rigorous validation and efficiency improvement cycles. Additional limitations of the current incarnation of Ranflood include the lack of an implemented detection routine and possible further refinements deriving from edge-case testing (cf. Section 2.4). Future studies can deepen these aspects related to edge cases and refine the implementation of Ranflood accordingly.

We deem Ranflood a useful tool to help users and system administrators in hampering the effects of ransomware attacks. Notably, Ranflood is a drop-in solution, similar to antivirus software. Hence, once equipped with easy-to-use graphical interfaces, it would be configured as a tool for the wider, low-technical-skills tier of computer users. We deem it important also to work towards integrating Ranflood in existing security suites. The potential wide audience of the tool and its contrasting action have the additional benefit of making ransomware attacks less lucrative and, thus, curtailing their attractiveness for malicious actors.

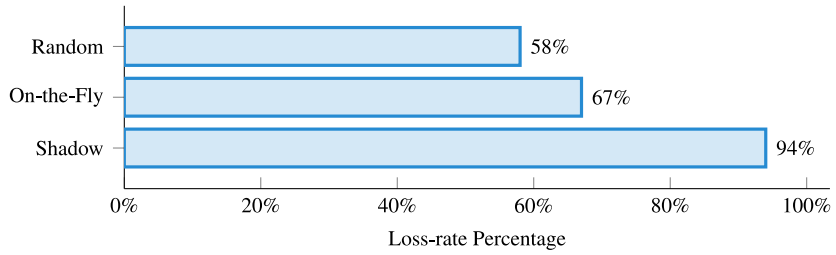

Fig. 2. Performance in terms of recovery-rate percentage—the higher the percentage the fewer files were lost to encryption—of the Random, On-the-Fly, and Shadow Ranflood flooding strategies against WannaCry; Ranflood activation delay of 30 s.

5. Conclusion

We focused on crypto-ransomware; the infamous category of malicious software characterised by the extortion of resources (usually, money) from users to regain control over their data which the ransomware encrypts. To contrast crypto-ransomware, we presented Ranflood. We discuss its position among alternatives, the strategies it offers to contrast different families of crypto-ransomware, and its implementation, particularly looking at its architecture—how it supports highly concurrent contrast operations and how its modularity allows Ranflood to host different (future) strategies. We presented the software functionalities provided to users and exemplified their usage and shown preliminary benchmarks with a concrete case, concerning the contrast against the WannaCry ransomware.

Future work on Ranflood includes providing new flooding strategies, e.g., using scattering routines to spread the content of the files of the user over different decoy files, which, when not encrypted, can support a distributed kind of restoration routine. Another important direction is implementing detection policies that allow Ranflood to automatically recognise possible threats and trigger its flooding mechanisms.

Research and development of Ranflood is an ongoing collaboration between Alma Mater Studiorum – Università di Bologna and Agenzia Regionale Prevenzione Ambiente e Energia (ARPAE) of Emilia-Romagna.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

We included a links to the code and all the data used in the experiments.

Acknowledgements

We thank Stefano Cattani for promoting and supporting the establishment of the collaboration between ARPAE and Università di Bologna. This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

References

[1] Liska A, Gallo T. Ransomware: Defending against digital extortion. O'Reilly Media, Inc.; 2016.
[2] Greengard S. The worsening state of ransomware. Commun ACM 2021;64(4):15–7.
[3] Perlroth N, Scott M, Frenkel S. Cyberattack hits Ukraine then spreads internationally. NY Times 2017. https://www.nytimes.com/2017/06/27/technology/ransomware-hackers.html.
[4] Joe C, Andres GL, Jill RS. Gas Stations Run Dry as Pipeline Races to Recover From Hacking - Bloomberg. 2021, Bloomberg, https://www.bloomberg.com/news/articles/2021-05-09/u-s-fuel-sellers-scramble-for-alternatives-to-hacked-pipeline.
[5] Person, Padraic Halpin CH. Irish Health Service hit by 'very sophisticated' ransomware attack. 2021, Reuters, https://www.reuters.com/technology/irish-health-service-hit-by-ransomware-attack-vaccine-rollout-unaffected-2021-05-14/.
[6] Abrams L. Ransomware attack hits Italy's Lazio region, affects COVID-19 site. 2021, BleepingComputer, https://www.bleepingcomputer.com/news/security/ransomware-attack-hits-italys-lazio-region-affects-covid-19-site/.
[7] Sheila A. M, Tracy P. M. WannaCry: Are your security tools up to date? Natl Law Rev 2017. https://www.natlawreview.com/article/wannacry-are-your-security-tools-to-date.
[8] Berardi D, Giallorenzo S, Melis A, Melloni S, Onori L, Prandini M. Data flooding against ransomware: Concepts and implementations. Comput Secur 2023;103295.
[9] Continella A, Guagnelli A, Zingaro G, Pasquale GD, Barenghi A, Zanero S, et al. Shieldfs: a self-healing, ransomware-aware filesystem. In: Schwab S, Robertson WK, Balzarotti D, editors. Proceedings of the 32nd annual conference on computer security applications. ACM; 2016, p. 336–47, URL http://dl.acm.org/citation.cfm?id=2991110.
[10] Gómez-Hernández JA, Álvarez-González L, García-Teodoro P. R-Locker: Thwarting ransomware action through a honeyfile-based approach. Comput Secur 2018;73:389–98.
[11] Lee S, Kim HK, Kim K. Ransomware protection using the moving target defense perspective. Comput Electr Eng 2019;78:288–99.
[12] Microsoft. Protect important folders with controlled folder access. 2022, https://docs.microsoft.com/en-us/microsoft-365/security/defender-endpoint/controlled-folders?view=o365-worldwide.
[13] Lee K, Yim K, Seo JT. Ransomware prevention technique using key backup. Concurr Comput: Pract Exper 2018;30(3):e4337.
[14] Rossow C, Dietrich CJ, Grier C, Kreibich C, Paxson V, Pohlmann N, et al. Prudent practices for designing malware experiments: Status quo and outlook. In: 2012 IEEE symposium on security and privacy. IEEE; 2012, p. 65–79.
[15] Connolly LY, Wall DS. The rise of crypto-ransomware in a changing cybercrime landscape: Taxonomising countermeasures. Comput Secur 2019;87:101568.
[16] Marsaglia G. Xorshift RNGs. J Stat Softw 2003;8(14):1–6.
[17] Tanenbaum A. Modern operating systems. Pearson Education, Inc.; 2009.
[18] Pyarali I, Harrison T, Schmidt DC, Jordan TD. Proactor - An object behavioral pattern for demultiplexing and dispatching handlers for asynchronous events. Tech. rep., Washington University; 1997.