applied sciences

MDPI

Article

Copy & Return: Evidence-Preserving Damage Suppression with Lightweight AI-Based Ransomware Screening

Jiseok Bang [1], Yohan Lee [1], Sang-su Lee [2], Yong-je Choi [2] and Seungkwang Lee [1,*]

[1] Department of Cyber Security, Dankook University, Yongin-si 16890, Gyeonggi-do, Republic of Korea; jsbang@dankook.ac.kr (J.B.); hanja1500@dankook.ac.kr (Y.L.)
[2] Cyber Security Research Division, Electronics and Telecommunications Research Institute (ETRI), Daejeon 34129, Republic of Korea; sangsu@etri.re.kr (S.-s.L.); choiyj@etri.re.kr (Y.-j.C.)
* Correspondence: sk.cryptographic@dankook.ac.kr

Abstract

Dynamic ransomware analysis presents a dilemma: terminating a sample prevents later behavior from being observed, whereas allowing destructive functions to complete can corrupt the analysis environment and its evidence. Copy & Return addresses this problem by suppressing selected destructive effects at function boundaries while allowing the original process to continue. An ABI-compatible substitute is introduced via dynamic-library interposition or binary injection to redirect a dangerous destination, preserve a mutable buffer, or bypass a destructive transformation before returning control to the unchanged caller. To identify potential intervention targets, we implement a static-analysis pipeline that ranks candidate functions and presents the underlying evidence for target confirmation. We evaluate candidate rankings on 10 selected ELF ransomware binaries and conduct detailed runtime case studies of MBRLocker, Conti, and Cylance. For MBRLocker, the 512-byte disk payload is redirected to a dummy file without modifying the actual boot sector. For Conti and Cylance, the original contents of eight target files per sample are preserved while file renaming, metadata appending, and ransom-note creation remain observable, demonstrating evidence-preserving damage suppression. Separately, we use numerical features obtained through static analysis to train and evaluate a lightweight ransomware screening model based on histogram-based gradient boosting. On 1156 binaries comprising 126 ransomware and 1030 benign samples, the model achieves a mean F1-score of 0.9124 and a mean false-positive rate of 1.62% under nested grouped cross-validation; the serialized model occupies approximately 39.3 KiB. These results demonstrate complementary roles for function-level damage suppression, static target-identification support, and lightweight AI-based ransomware screening.

Keywords: ransomware analysis; function substitution; dynamic interposition; binary injection; non-destructive analysis; ransomware detection

Check for updates

Academic Editor: Christos Bouras

Received: 6 August 2026
Revised: 24 September 2026
Accepted: 30 September 2026
Published: 5 October 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Ransomware analysis must balance behavioral visibility against damage. Empirical and survey studies have characterized the attack lifecycle, encryption behavior, and the main families of detection, prevention, and recovery defenses [1,2]. Static analysis can expose control flow and constants without running a sample, but adversarial code transformations and path explosion can make such analysis incomplete or computationally impractical [3]. Dynamic analysis observes interactions with files, processes, and libraries,

but safe execution normally requires a sandbox or other controlled environment [4]. Ransomware makes this tradeoff unusually sharp because a short observation interval can include destructive disk writes or irreversible file encryption. UNVEIL demonstrated large-scale behavioral detection by monitoring file-system and desktop changes [5], whereas Redemption coupled endpoint monitoring with restoration [6]. Copy & Return addresses a complementary question: can the specific destructive operation be replaced while the original process remains alive long enough to reveal later behavior?

The method works at a compiled-function boundary. First, a target function, such as a disk-opening routine or an encryption function, is identified. Second, the function prototype, argument locations, buffer aliasing, and return convention are recovered. Third, an application binary interface (ABI)-compatible substitute is installed through dynamic-library interposition or static binary injection. Finally, the substitute redirects a dangerous destination, copies plaintext instead of generating ciphertext, or returns without modifying an in-place buffer. Control then returns to the original caller. The caller-visible contract is preserved sufficiently for the process to continue, but the destructive effect is removed.

Identifying a suitable target requires analysis of each sample. Our static analyzer combines information from multiple analysis tools and ranks candidate functions using the same set of rules. Each candidate is accompanied by symbol, instruction, and call-site evidence for review. The ranking guides the review of potential intervention sites, while target confirmation and ABI recovery remain necessary.

We report candidate identification and ranking results for 10 selected ELF ransomware binaries, with detailed case studies of MBRLocker, Conti, and Cylance. The case studies draw on payloads, file metadata, screenshots, binary-analysis records, and before/after file measurements. Their results illustrate the distinction between containment and termination: selected malicious effects are suppressed while subsequent behavior remains observable.

We further examine whether numerical static-analysis features can support a compact AI-based ransomware screening model suitable for lightweight deployment scenarios. We train and evaluate a histogram-based gradient boosting (HGB) model to distinguish ransomware from benign binaries.

These components address complementary stages of the analysis workflow: the AI model provides binary-level ransomware screening, the static-analysis pipeline prioritizes potential intervention targets, and Copy & Return suppresses a selected destructive effect at a confirmed function boundary.

The contributions of this paper are:

- We present Copy & Return, an ABI-preserving function-substitution mechanism that suppresses selected destructive effects while returning control to the unchanged caller through dynamic interposition or binary injection.
- We develop a static-analysis pipeline that consolidates function-level evidence from multiple analysis tools and ranks candidate intervention targets, and we evaluate its rankings on 10 selected ELF ransomware binaries.
- We evaluate evidence-preserving damage suppression through detailed runtime case studies of MBRLocker, Conti, and Cylance, showing that selected destructive effects can be suppressed while subsequent behaviors—including file renaming, metadata appending, ransom-note creation, and payload retention—remain observable.
- We train and evaluate a lightweight AI-based ransomware screening model using numerical static-analysis features. On 1156 binaries, the HGB model achieves a mean F1-score of 0.9124 and a mean false-positive rate of 1.62% under nested grouped cross-validation, with a serialized model size of approximately 39.3 KiB.

2. Background and Related Work

2.1. Static, Dynamic, and Non-Destructive Analysis

Static malware analysis scales without executing a sample, but adversarial code transformations and path explosion can make its conclusions incomplete or computationally impractical [3]. Automated dynamic systems record concrete API and system behavior, while hybrid systems combine runtime code discovery with static analysis to recover packed or self-modifying regions [4,7,8]. Their observations still depend on the environment and the paths reached during execution. Malware can recognize emulators and alter its behavior, and Linux malware studies confirm that packing, evasion, and anti-analysis techniques are not confined to Windows samples [9,10].

Ransomware-specific systems approach the safety problem at several levels. UNVEIL monitors file-system, desktop, and input/output (I/O) behavior to distinguish ransomware executions [5]. Redemption performs real-time endpoint protection and recovery [6]. CryptoLock monitors user-data transformations to stop suspicious processes, while ShieldFS adds ransomware-aware monitoring and self-healing at the file-system layer [11,12]. PayBreak escrows symmetric key material for recovery, and a later storage defense prevents maliciously encrypted page-cache data from being committed to persistent media [13,14]. Copy & Return occupies a different point in this design space: it replaces a selected operation before the destructive effect occurs and lets the process continue. Its principal analytical benefit is therefore post-intervention visibility.

Figure 1 summarizes an illustrative ransomware execution workflow [2]. The numbers denote (1) infiltration, when the malware reaches the target system; (2) execution, when the payload starts running; (3) command and control (C2) communication, when present, to exchange configuration or key-related information; (4) scanning and target identification, to locate files or other resources; (5) encryption, which transforms file contents in cryptographic ransomware; and (6) extortion, in which a ransom demand is presented. These stages are schematic and need not occur in every family or in a fixed order. Copy & Return targets a confirmed destructive operation, such as a disk write or a file-content transformation, and suppresses its selected effect while allowing subsequent behavior to remain observable.

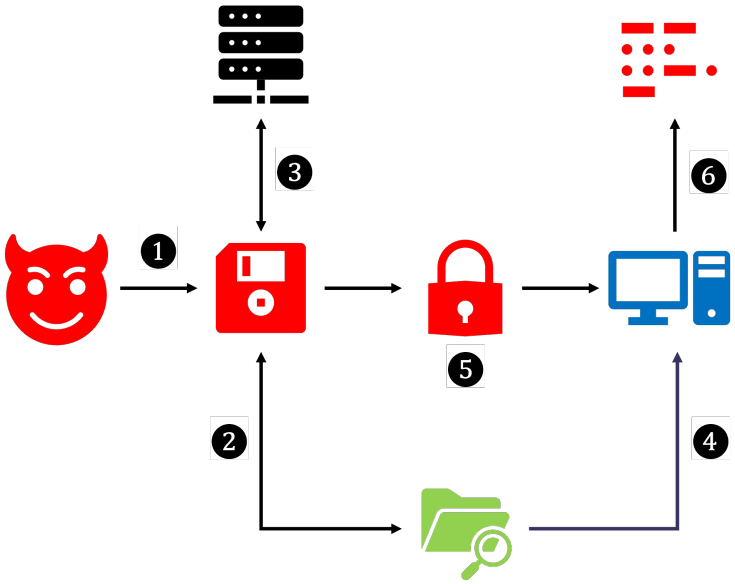


Figure 1. Ransomware execution stages considered by Copy & Return. The intervention point lies immediately before the selected destructive operation, allowing later stages to remain observable.

2.2. Function Interposition and Binary Rewriting

Runtime interposition changes which implementation satisfies a dynamically resolved symbol. On Executable and Linkable Format (ELF) systems, a preloaded shared object can be searched before the ordinary provider, allowing a compatible wrapper to replace or mediate the call [15]. This route is effective when the target function is dynamically linked, and its symbol or call interface is known.

User-defined and statically linked functions require a binary-level route. General binary-instrumentation and reassembly systems demonstrate that compiled code can be modified without source [16,17]. Copy & Return uses the same broad capability for a narrow safety goal: redirect a call site to a small ABI-compatible substitute and then return to the unmodified caller.

Finding the correct cryptographic boundary is itself a studied reverse-engineering problem. ReFormat separates decryption processing from later protocol handling and identifies the lifetime of plaintext buffers [18]. Dynamic heuristics can identify cryptographic primitives and extract keys from binary executions, while CIS combines multiple heuristics to localize cryptographic functions in malware [19,20]. Static data-flow-graph isomorphism supports identification across compiler variants, and Where's Crypto? extends this direction toward unknown proprietary primitives [21,22]. These techniques complement rather than replace Copy & Return: they can help locate and characterize a target whose recovered ABI is then used to synthesize the substitute. Our static analyzer supports this target-identification stage by combining recovered function-level evidence from multiple analysis tools and applying common criteria to rank candidate intervention sites. Candidate functions include cryptographic routines and resource-access functions potentially involved in malicious behavior. The resulting records support analyst review of potential targets, while the caller-visible contract must still be recovered before a compatible substitute can be constructed.

2.3. Ransomware Detection and Platform-Specific Defenses

Ransomware is not limited to desktop hosts. Android task hijacking can be used to lock device tasks and implement ransomware-like denial of service [23]. HelDroid detects mobile-ransomware building blocks such as unsolicited locking, encryption, and ransom demands, while KRDroid models Android ransomware through behavior patterns [24,25]. Longitudinal telemetry also shows that potentially harmful Android applications can persist after detection, partly because ordinary security applications have limited removal authority [26]. On Internet of Things (IoT) infrastructure, SPOT used bare-metal network-attached storage (NAS) devices to observe ransomware attacks and executed samples in a controlled NAS sandbox [27].

Feature-based detection and policy enforcement address related but distinct tasks. API-RDS derives static API-package features and trains a predictive classifier [28]. RansomDroid applies unsupervised learning to Android ransomware detection [29]. RansomCare combines lightweight static and dynamic analysis with data protection on smartphones [30]. Aurasium shows that application repackaging and user-level API interposition can support policy enforcement on Android, although platform constraints and signing implications remain [31]. In this work, the AI-based screening model distinguishes ransomware from benign binaries using numerical static-analysis features, whereas Copy & Return replaces an identified function to suppress its destructive effect while retaining subsequent behavior.

3. Materials and Methods

3.1. Problem Definition and Scope

3.1.1. Analysis Problem

Let a suspicious process invoke a target function as in Equation (1):

$$y = f(a_1, a_2, \ldots, a_n), \tag{1}$$

where one or more arguments identify a destructive destination or mutable buffer. Executing f normally may damage the host. Terminating the process before f prevents the damage but also suppresses all later events. The objective is to construct a substitute g such that:

1. g obeys the caller-visible ABI of f;
2. g removes or redirects the destructive effect;
3. g returns a value compatible with the caller's expected control flow;
4. The process continues sufficiently to expose later behavior.

Copy & Return does not require g to be semantically equivalent to f. It requires analysis-compatible continuity: the protected state remains safe, while enough of the malicious control flow continues to generate evidence.

3.1.2. Threat and Trust Boundary

The evaluated setting is a controlled x86-64 Linux analysis host. The analyst can inspect and transform the ELF executable and can control the process launch environment. The target is a user-space function or dynamically linked library function whose ABI can be recovered. The method does not claim protection when malware has already compromised the kernel, performs unmediated hardware access outside the intercepted path, detects and bypasses the substitute, or deliberately corrupts the caller after returning. The three case studies evaluate containment and behavioral continuity. The additional static-analysis and classification experiments evaluate target-identification support and ransomware screening, respectively.

3.1.3. Target Identification and Ransomware Screening

The static analyzer supports target identification by ranking candidate functions and presenting the available evidence. Candidates may include cryptographic routines or resource-access functions potentially involved in malicious behavior. The analyst reviews these records, confirms the target, and recovers its caller-visible contract before applying the original Copy & Return procedure.

A separate classifier uses numerical static-analysis features to distinguish ransomware from benign binaries. Its output is a binary-level screening result rather than a function-level intervention decision. Copy & Return remains detector-agnostic: classification and candidate ranking support the analysis workflow, while the containment mechanism operates at an already confirmed function boundary. A classification result alone does not determine which function should be replaced or which substitute is compatible with its caller.

3.1.4. Research Questions

We consider five research questions (RQs):

RQ1— **Neutralization.** Can redirect, copy, and immediate-return substitutes prevent the selected destructive disk or encryption effect?

RQ2— **Continuity.** Does the ransomware continue with observable post-intervention behavior?

RQ3— **Evidence.** Do the retained payload, file metadata, appended material, and screenshots consistently support the neutralization and continuity claims?

RQ4— **Target Identification.** To what extent can static analysis support the identification and prioritization of candidate intervention sites?

RQ5— **AI-Based Screening.** How effectively can an AI-based model distinguish ransomware from benign binaries using numerical static-analysis features?

3.2. Copy & Return Method

3.2.1. Pipeline

The method comprises five stages:

1. Identify a suspicious function or resource access;
2. Recover the prototype, calling convention, buffers, sizes, return type, and aliasing;
3. Synthesize an ABI-compatible substitute;
4. Install the substitute through dynamic interposition or binary injection;
5. Execute a safety policy and return to the original caller.

Figure 2 illustrates how Copy & Return redirects library and user-defined calls to compatible substitutes. The solid paths represent ordinary function calls. For an unresolved library call under lazy binding, labels (1)–(5) indicate entry into the procedure linkage table (PLT), lookup in the global offset table (GOT), return to the binding stub, transfer to the common resolver entry, and symbol resolution by the runtime linker, respectively. Label (6) represents subsequent calls through the resolved GOT entry, while (7) represents a direct call to a user-defined function. The dashed paths labeled (1) and (7) show the corresponding calls redirected to an interposed library substitute and an injected substitute, respectively. In both cases, the substitute must preserve the caller-visible ABI and expected return behavior.

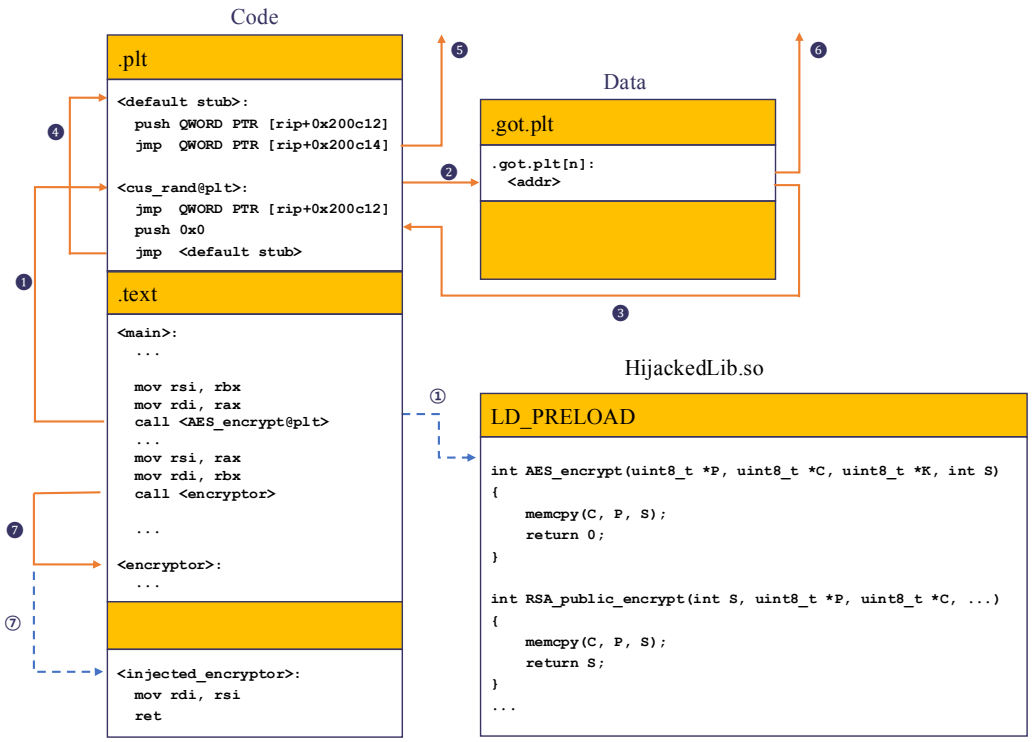


Figure 2. Copy & Return method. Solid lines represent normal function calls, and dashed lines represent redirected calls through an injected or interposed substitute.

3.2.2. Identifying the Target

The analyst first identifies a candidate intervention boundary. For a boot locker, the boundary may be a write to `/dev/sda`. For crypto-ransomware, the usual data-flow pattern is file read, transformation, and file write. A function called between the read and write is a candidate encryption routine when it repeatedly reads and writes a buffer, uses bitwise operations, consumes a size parameter, or receives the read buffer as an argument.

Symbols simplify this step, but stripped binaries require register and memory analysis. Under the System V x86-64 ABI, the first integer or pointer arguments are passed in `RDI`, `RSI`, `RDX`, `RCX`, `R8`, and `R9`. Tracking values from a preceding `read` call can therefore distinguish plaintext, ciphertext, context, key, and size arguments.

Table 1 summarizes the replacement cases. Here, E denotes an encryption operation; P and C are input and output buffers, B is an in-place buffer, K is a key argument, and S is the byte count.

Table 1. Target-function forms and Copy & Return policies.

Form	Example Contract	Substitute Policy
Out-of-place	$E(P, C, K, S)$	Copy S bytes from P to C; return success
In-place	$E(B, K, S)$ or $E(B, S)$	Return without modifying B
Resource open	open$(path, mode)$	Replace the target resource path with a dummy path

3.2.3. Automated Candidate Generation and Ranking

The static analyzer supports the first stage of Copy & Return by organizing recovered functions and their supporting evidence into a ranked candidate list. It does not change the subsequent requirements for target confirmation, ABI recovery, or compatible substitution.

Structural Analysis and Evidence Extraction

An in-house analyzer, Ghidra, and angr perform structural analysis in parallel. The in-house analyzer uses executable metadata and disassembled instructions to recover candidate entries and call-related evidence. Ghidra supplies recovered functions, external symbols, and call relationships [32]. The angr adapter uses static CFGFast analysis to recover functions and control-flow information [33]. The recovered function records are combined, and capa capability evidence is then associated with the corresponding code locations [34]. In this integration, capa supplies capability annotations rather than a fourth independent function-recovery result.

Combining Function Records

Records are associated with the sample SHA-256 and normalized code locations. Within the same binary, internal function records with the same relative virtual address are merged while retaining their source tools and reported ranges. External symbols are identified using the available library, symbol, and version information. Symbols from different tools are not forced into a common identity when the library is unknown. Overlapping code ranges alone are not sufficient to merge different entry points. Boundary disagreements and incomplete tool results remain visible in the resulting records.

Common Evidence Rules

The same rules are applied to the consolidated records rather than summing tool-specific relevance scores. The rules examine cryptographic names, capabilities mapped to

specific code locations, instruction patterns, resource-access APIs, and explicitly supported data-flow relations. For a candidate f, the review score is given by Equation (2):

$$S(f) = \sum_{j=1}^{m} w_j I_j(f), \tag{2}$$

where m is the number of evidence rules, w_j is the weight of rule j, and $I_j(f) = 1$ when the available evidence satisfies rule j, and $I_j(f) = 0$ otherwise. Each rule contributes at most once, even when several tools report supporting evidence. Missing evidence is retained as unknown rather than interpreted as proof that a behavior is absent.

Table 2 lists the rule weights. The weights determine how much each evidence rule contributes to the score. The resulting score indicates review priority, not the probability that a function is malicious or can be safely replaced.

Table 2. Evidence rules used for candidate prioritization.

Supported Evidence	Weight
Cryptographic symbol-name match	20
Localized cryptographic capability	20
Buffer-transformation instruction pattern	8
Presence of bitwise operations	2
Resource-access API match	8
Explicitly verified static read–transform–write relation	20
Localized behavior-related capability	8

The buffer-transformation rule requires memory-read and memory-write instructions, at least four bitwise instructions, and either a rotation or a backward branch in the reported instruction statistics. This pattern is a review cue rather than proof of cryptographic semantics. Likewise, the read–transform–write rule requires an explicitly verified static relation; the presence of file-related calls alone does not satisfy it.

Candidate Ranking

All recovered function records remain in the candidate list, including those that match none of the evidence rules. Candidates are ranked by decreasing $S(f)$, with ties resolved by the canonical candidate identifier in lexicographic order. No score-based exclusion threshold is applied. The ranking indicates review priority; it does not confirm that a function is destructive or suitable for replacement.

For each candidate, the output records its location, available names, supporting evidence, and the tools that supplied that evidence. Function locations reported in public analyses are used only for evaluation, not to guide candidate generation.

3.2.4. Dynamic Interposition

For a dynamically linked function, a preloaded provider exports the same symbol and prototype. If a suspicious path equals `/dev/sda`, the wrapper replaces it with `/home/dummy/MBRLocker` before calling the ordinary file-opening implementation. For an out-of-place cipher with plaintext P, ciphertext C, and length S, the replacement applies the buffer-copy operation in Equation (3):

$$C[0:S] \leftarrow P[0:S] \tag{3}$$

and returns a success-compatible value. This keeps the output buffer initialized without creating ciphertext.

The same route was used to replace Conti's Rivest–Shamir–Adleman (RSA) routine, `RSA_public_encrypt`. The substitute copied the source material to the destination and returned the source length. Consequently, file-key material that would ordinarily be public-key encrypted remained observable in the appended region.

3.2.5. Binary Injection

When the target is user-defined or statically linked, the call site is redirected to injected instructions. The exact substitute depends on the recovered ABI:

- If the source and destination are distinct, copy the requested byte count and return;
- If the encryption is in-place, return immediately;
- If a caller checks a result code, place the expected result in the return register before returning.

An incorrect argument order, buffer length, alias decision, or return code can crash the process or change its path. The method therefore assumes that target discovery and ABI recovery have already been completed by the analyst or an upstream component.

3.2.6. Analysis Support and Ransomware Screening

Figure 3 shows how static analysis supports the original Copy & Return procedure. Structural recovery and capability evidence produce a ranked list for analyst review. After the target and its caller-visible contract have been confirmed, the original interposition or injection mechanism applies without changing its substitution policies or return-to-caller behavior.

Ransomware screening is evaluated separately using numerical static-analysis features. A histogram-based gradient boosting classifier produces a binary-level decision, whereas candidate ranking addresses the location of a potential intervention. The classification experiment uses previously extracted feature records and is not an evaluation of an automatic runtime connection between the classifier and Copy & Return.

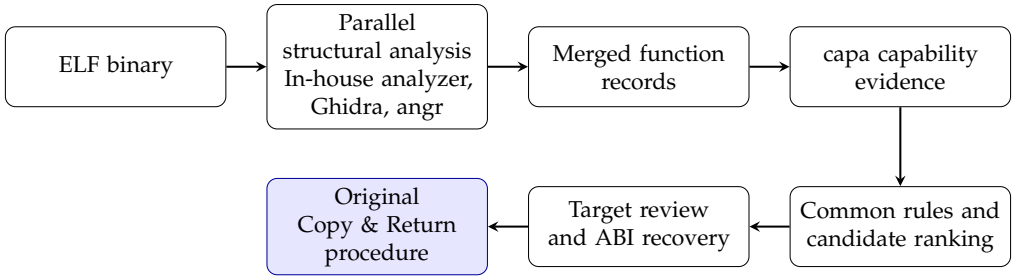


Figure 3. Static-analysis support for Copy & Return. Parallel structural analysis produces function records, which are combined, linked to capability evidence, and ranked using common rules. The blue node denotes the original containment procedure, applied after target confirmation and ABI recovery.

3.3. Experimental Materials

3.3.1. Evidence Provenance and Environment

The experiments were conducted on an Intel Core i9 host with 32 GB of memory running Ubuntu 22.04 Long-Term Support (LTS). The runtime-containment evaluation covers three ransomware samples and records the intervention point, safety policy, and post-intervention artifacts for each case. The samples were obtained from MalwareBazaar (https://bazaar.abuse.ch, accessed on 25 May 2026) and can be located using the SHA-256 hashes reported in Table 3, subject to the repository's access policies. Ransomware binaries are not redistributed with this paper.

Table 3 lists the 10 selected ELF ransomware binaries used in the case studies and static-analysis evaluation, comprising the three case-study samples and seven additional samples. SHA-256 hashes identify the specific binaries, and the sample labels should not be generalized to every variant sharing a family name.

In the runtime case studies, MBRLocker targets the first 512 bytes of a disk. Conti uses the user-defined `ECRYPT_encrypt_bytes` routine for in-place content transformation and calls `RSA_public_encrypt` for key material. Cylance calls an in-place `ChaCha_XOR` routine. The intervention policies and resulting observations are described in the corresponding case studies.

Table 3. Selected ransomware samples used in the case studies and static-analysis evaluation.

Sample	SHA-256
MBRLocker	`8d4bafbdb15653fe34ae9bc31fc0d4b3340a44ca541505850f063bebc218a8fc`
Conti	`8b57e96e90cd95fc2ba421204b482005fe41c28f506730b6148bcef8316a3201`
Cylance	`d1ba6260e2c6bf82be1d6815e19a1128aa0880f162a0691f667061c8fe8f1b2c`
Black Basta	`0d6c3de5aebbbe85939d7588150edf7b7bdc712fceb6a83d79e65b6f79bfc2ef`
ESXiArgs	`11b1b2375d9d840912cfd1f0d0d04d93ed0cddb0ae4ddb550a5b62cd044d6b66`
Lockton	`cf37b3caea9b8f6bf3297f2d9c562703ad0eb4f6a27df9b105bd7c9aa9ca95ee`
Erebus	`0b7996bca486575be15e68dba7cbd802b1e5f90436ba23f802da66292c8a055f`
Payload ESXi	`bed8d1752a12e5681412efbb8283910857f7c5c431c2d73f9bbc5b379047a316`
Royal	`06abc46d5dbd012b170c97d142c6b679183159197e9d3f6a76ba5e5abf999725`
RansomEXX	`cb408d45762a628872fa782109e8fcfc3a5bf456074b007de21e9331bb3c5849`

3.3.2. Target Files and Observations

The crypto-ransomware experiment used eight files: three PNG images, one PowerPoint presentation, one PDF report, one spreadsheet, and two text files. The recorded observations are:

1. File name, size, and inode before and after execution;
2. Original-file readability after restoring the extension;
3. Ransomware-specific extension changes;
4. Creation of ransom-note files;
5. Appended material left by the sample.

These runtime experiments isolate the downstream containment question: whether replacing the identified destructive routine preserves file content while allowing later ransomware behavior to remain observable.

3.3.3. AI-Based Ransomware Screening

The ransomware-screening experiment uses 1156 unique binaries, comprising 126 ransomware binaries in 41 family groups and 1030 benign binaries in 505 source groups. Duplicate sample hashes are excluded. This dataset is used for binary-level ransomware classification.

The model uses 45 numerical features describing each binary and its static-analysis results. These features were extracted by the earlier static analyzer, not by the multi-tool candidate-ranking pipeline described above. All six trace-related features are zero throughout this static-only dataset. No sample execution is required for the classification experiment.

We evaluate histogram-based gradient boosting (HGB) alongside XGBoost, CatBoost, and a weighted ensemble. Evaluation uses five-fold grouped cross-validation repeated three times. Ransomware-family and benign-source groups do not overlap between training and test partitions. Model hyperparameters and classification thresholds are selected using three-fold grouped validation within each outer training partition; ensemble weights are selected within the same inner validation procedure.

For each repetition, we combine the complete out-of-fold predictions and calculate precision, recall, F1-score, and false-positive rate, treating ransomware as the positive class. We report the mean of each metric across the three repetitions; any reported standard deviation describes variation across these repetitions. The false-positive rate is calculated over benign binaries. These measurements evaluate binary-level ransomware screening, not function-level localization or runtime containment.

4. Results

We first present the runtime results for MBRLocker, Conti, and Cylance, followed by the static target-identification results and the AI-based ransomware-screening experiment.

4.1. MBRLocker: Redirection and Payload Capture

The MBRLocker sample prepares a 512-byte payload and attempts to write it to `/dev/sda` before invoking a reboot command. The interposed file-opening function substitutes a dummy path, so the subsequent write captures the payload without modifying the actual boot sector. Figure 4 shows the visual payload associated with the sample.

The captured sector ends with the valid MBR marker `55 AA`. Table 4 summarizes its layout. Capturing the full sector is analytically preferable to merely blocking the open call because it retains the complete disk-write payload for inspection.

Table 4. Compact layout of the redirected 512-byte Master Boot Record (MBR) payload.

Offset	Size	Observed Role
`0x000-0x117`	280 B	16-bit bootstrap
`0x118-0x1BD`	166 B	Inline run-length-encoded pixel stream
`0x1BE-0x1FD`	64 B	Zeroed partition table
`0x1FE-0x1FF`	2 B	Signature `55 AA`

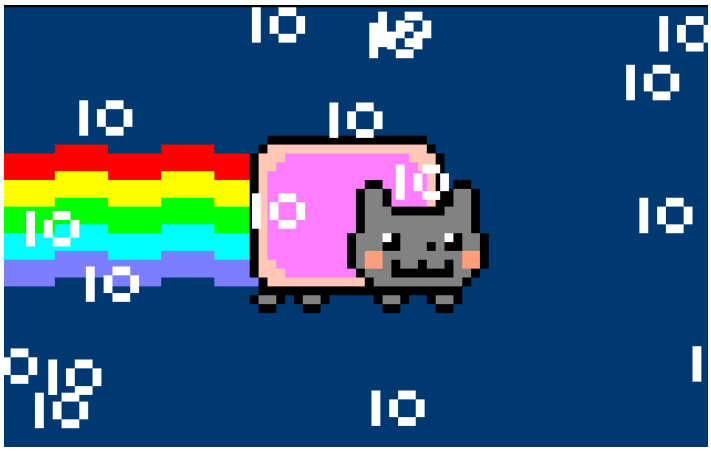


Figure 4. MBRLocker artifact: the Nyan Cat display encoded in the redirected 512-byte boot-sector payload.

4.2. Conti: In-Place Return and Key-Material Exposure

Assembly data flow showed that the buffer passed to `read` was used as both the plaintext and ciphertext argument of `ECRYPT_encrypt_bytes`. The experiment redirected that call to an immediate return. A second substitute replaced `RSA_public_encrypt` with a source-to-destination copy. The files therefore retained their original bodies, while the ransomware continued to rename them and append 512 bytes.

Table 5 reports the results for all eight files. Every inode remains unchanged, every name gains `.conti`, and every file gains exactly 512 bytes. The sample also creates `CONTI_README.txt`.

Table 5. Before/after records for the Conti experiment.

Before			After			Difference
Name	**Bytes**	**Inode**	**Name**	**Bytes**	**Inode**	**Bytes**
f12.png	89,860	1053765	f12.png.conti	90,372	1053765	512
flags.png	113,754	1053766	flags.png.conti	114,266	1053766	512
maps.png	2,720,902	1053767	maps.png.conti	2,721,414	1053767	512
smp_present.pptx	3,218,946	1053770	smp_present.pptx.conti	3,219,458	1053770	512
smp_report.pdf	279,844	1053771	smp_report.pdf.conti	280,356	1053771	512
smp_sheets.xlsx	8767	1053772	smp_sheets.xlsx.conti	9279	1053772	512
test2.txt	78	1053768	test2.txt.conti	590	1053768	512
test.txt	450	1053769	test.txt.conti	962	1053769	512
	—		CONTI_README.txt	1480	1053774	—

For `smp_report.pdf`, the appended region contains a 32-byte key, 12-byte initialization vector (IV), and 8-byte counter:

- key: `70F06D18 4EB05402 E990FEA4 2514BA1A 06F7C42D 5A20476C C65232CD 38332CAA`;
- IV: `F1E9775E F496B7AA 64000000`;
- counter: `11000000 30CE0000`.

The remaining 460 bytes are zero. This pattern supports the interpretation that the public-key wrapping operation was bypassed: the key-related fields remain directly observable instead of being replaced by a full RSA ciphertext.

4.3. Cylance: In-Place ChaCha Bypass

For the Cylance sample, both the input and output registers supplied to `ChaCha_XOR` referenced the same buffer, and that buffer was subsequently written to the target file. Redirecting the call to an immediate return preserved the buffer. Table 6 reports the results: all eight files retain their inodes, receive the `.Cylance` extension, and gain exactly 64 bytes. The sample also creates `CYLANCE_README.txt`.

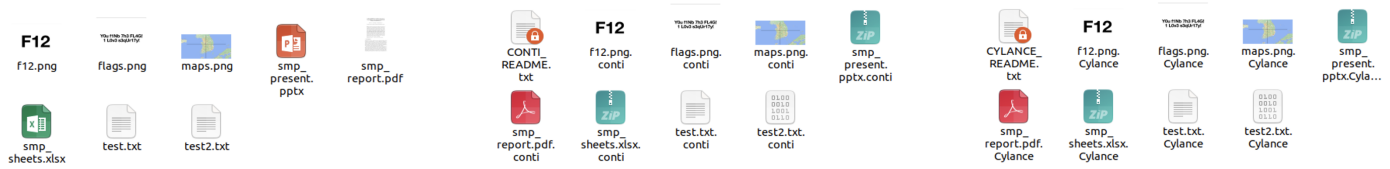
Table 6. Before/after records for the Cylance experiment.

Before			After			Difference
Name	**Bytes**	**Inode**	**Name**	**Bytes**	**Inode**	**Bytes**
f12.png	89,860	1053756	f12.png.Cylance	89,924	1053756	64
flags.png	113,754	1053757	flags.png.Cylance	113,818	1053757	64
maps.png	2,720,902	1053758	maps.png.Cylance	2,720,966	1053758	64
smp_present.pptx	3,218,946	1053761	smp_present.pptx.Cylance	3,219,010	1053761	64
smp_report.pdf	279,844	1053762	smp_report.pdf.Cylance	279,908	1053762	64
smp_sheets.xlsx	8767	1053763	smp_sheets.xlsx.Cylance	8831	1053763	64
test2.txt	78	1053759	test2.txt.Cylance	142	1053759	64
test.txt	450	1053760	test.txt.Cylance	514	1053760	64
	—		CYLANCE_README.txt	1369	1053776	—

4.4. Visual and Cross-Case Evidence

Figure 5 shows the file-browser screenshots. The post-execution views preserve the renamed target set and newly created ransom notes. Preview behavior can be affected by the changed extensions, so the screenshots alone are not used as proof of byte-level integrity; the size, inode, appended-material, and readability records provide the complementary evidence.

Table 7 summarizes the cross-case result. The selected primary effects are neutralized, but each sample still exposes a later artifact useful for analysis.



(**a**) Original target files (**b**) After Conti (**c**) After Cylance

Figure 5. Screenshot evidence showing the target files before execution and the renamed files and ransom notes after the Conti and Cylance runs.

Table 7. Cross-case summary of the experimental artifacts from the three runtime experiments.

Sample	Selected Function/Effect	Copy & Return Action	Protected Outcome	Retained Later Evidence
MBRLocker	Open and write system disk	Redirect disk path	Actual boot sector unchanged	Complete 512-byte payload captured
Conti	In-place content cipher and RSA key wrapping	Return; copy source to destination	Eight file bodies retained	Eight renames, 512-byte append per file, key material, ransom note
Cylance	In-place `ChaCha_XOR`	Return	Eight file bodies retained	Eight renames, 64-byte append per file, ransom note

4.5. Static Target Identification

Table 8 presents example reference functions and their observed ranks for the 10 selected samples listed in Table 3. The MBRLocker and Conti entries use the earlier native-analyzer records; the remaining entries use the multi-tool analysis records. The table illustrates individual cases rather than reporting an aggregate success rate over the selected samples.

The earlier analyzer identifies only one candidate for MBRLocker, `fopen`. Its rank therefore does not show how well the analyzer distinguishes among multiple functions. In the Conti record, `RSA_public_encrypt` appears at rank 6, whereas the content-transformation routine `ECRYPT_encrypt_bytes` appears at rank 11. In the multi-tool Cylance record, `ChaCha_XOR` appears at rank 4.

Table 8. Example reference functions and observed ranks in the selected static-analysis cases.

Sample	Reference Function or Operation	Rank(s)
MBRLocker	`fopen`	1
Conti	`ECRYPT_encrypt_bytes`; `RSA_public_encrypt`	11; 6
Cylance	`encrypt_file`; `ChaCha_XOR`	2; 4
Black Basta	`EncryptFileBytes`; `EncryptionThread`	1; 5
ESXiArgs	`encrypt_simple`; `sosemanuk_encrypt`; `encrypt_file`	1; 3; 6
Lockton	`encrypt_file`	3
Erebus	`elib_fw_scan_dir_result_encrypt`	10
Payload ESXi	SSE2 and AVX2 stream-transformation routines	2; 3
Royal	`AES_encrypt`; `encrypt`	14; 299
RansomEXX	`CryptOneBlock`; `ProcessFileHandleWithLogic`	20; 152

Ranks follow the order of the listed functions or operations.

For several additional samples, the analyzer ranks documented transformation or file-processing routines near the top of the candidate list. However, this pattern is not consistent across all selected cases: Royal's `AES_encrypt` appears at rank 14, and RansomEXX's `CryptOneBlock` appears at rank 20. These results show how ranking can guide target review, but a fixed top-10 cutoff may exclude relevant functions. Matching a documented function does not by itself establish that a substitute preserves its caller-visible contract.

4.6. AI-Based Ransomware Screening Performance

Table 9 summarizes the classification results on the dataset of 1156 unique binaries. HGB achieves the highest mean F1-score among the individual classifiers, with precision of 0.8784, recall of 0.9497, and a false-positive rate of 1.62%. The weighted ensemble obtains a slightly higher mean F1-score of 0.9150, compared with 0.9124 for HGB.

Table 9. Ransomware classification under nested grouped cross-validation.

Model	Precision	Recall	F1-Score	FPR (%)
XGBoost	0.8083	0.7143	0.7583 ± 0.0128	2.07
CatBoost	0.8032	0.7884	0.7957 ± 0.0122	2.36
HGB	0.8784	0.9497	0.9124 ± 0.0080	1.62
Ensemble	0.8808	0.9524	0.9150 ± 0.0086	1.59

Values are means across three complete out-of-fold repetitions. For F1-score, $\pm$ denotes the population standard deviation across repetitions. Ransomware-family and benign-source groups are separated between training and test partitions.

The serialized HGB model occupies 40,214 bytes, approximately 39.3 KiB, excluding the feature extractor and inference runtime. This compact AI-based model screens binaries using the previously extracted numerical static-analysis features. We therefore present HGB as a compact single-model option rather than the highest-scoring configuration overall.

The classification results address whether a binary is labeled as ransomware, whereas the target-identification results address where an analyst should inspect the binary. Neither result extends the demonstrated runtime containment effects beyond the three original case studies.

5. Discussion

5.1. Answers to the Research Questions

RQ1— **Neutralization.** The redirected MBR write preserves the actual boot sector while capturing the full 512-byte payload. The immediate-return and copy policies preserve the original bodies of the Conti and Cylance target files.

RQ2— **Continuity.** The two crypto-ransomware samples continue far enough to rename every target, append family-specific material, and create ransom notes. MBRLocker's payload remains available for analysis. These artifacts support the central claim that intervention need not erase the remaining execution context.

RQ3— **Evidence.** The full before/after tables show stable inodes and exact per-family size increments across all eight files. The Conti key, IV, counter, and zero-padded region support the key-wrapping bypass interpretation. The screenshots independently show the renamed sets and note creation. No single artifact is treated as sufficient on its own.

RQ4— **Target Identification.** Static analysis provides candidate functions, supporting evidence, and review priorities. As shown in Table 8, 8 of the 10 selected binaries have at least one listed reference function or operation among the top 10 candidates. The highest-ranked listed references for Royal and RansomEXX appear at ranks 14 and 20, respectively. These selected cases illustrate how ranking can guide target review, while showing that a fixed top-10 cutoff may exclude relevant functions. They do not establish automatic target confirmation.

RQ5— **AI-Based Screening.** The HGB model achieves a mean F1-score of 0.9124 and a mean false-positive rate of 1.62% under nested grouped cross-validation. Its serialized size is approximately 39.3 KiB, excluding the feature extractor and inference runtime. These results support binary-level ransomware screening

with a compact model; they do not demonstrate automatic selection or execution of a compatible substitute.

5.2. Limitations

First, runtime containment is evaluated through three case studies, with eight target files used in each crypto-ransomware experiment. These results demonstrate case-specific feasibility rather than general protection across ransomware families.

Second, target identification and ABI recovery require analyst knowledge. A wrong prototype, alias decision, or return value can terminate or perturb the sample.

Third, intervention is limited to the identified function boundaries and does not cover execution paths that bypass them.

Fourth, the classifier is evaluated using previously extracted numerical features, separately from the multi-tool ranking pipeline. Group-disjoint evaluation limits family and source overlap between training and test partitions, but does not replace independent external validation. Classification false-positive rates were measured; the compatibility of benign programs under actual intervention policies was not. The reported 39.3 KiB size covers only the serialized HGB model; it excludes feature extraction and the inference runtime. End-to-end memory use, latency, and energy consumption on resource-constrained devices remain to be evaluated.

Finally, end-to-end detector-guided containment has not been evaluated. Detector error propagation, alert-to-policy latency, and integration across diverse devices require further evaluation.

5.3. Future Work

Future work will evaluate the ransomware-screening model on independent datasets and resource-constrained platforms. We will measure feature-extraction cost, inference latency, memory use, and energy consumption to assess whether the compact model supports lightweight deployment.

We will also integrate ransomware screening, target analysis, and Copy & Return, examining how detection errors affect subsequent analysis and containment. This evaluation will include target confirmation and the preservation of benign program behavior under intervention.

Another direction is to use AI models from the initial stages of static analysis, rather than only to review previously generated candidates. By analyzing instruction sequences, data-flow relations, and calling contexts, these models may support function recovery, semantic interpretation, candidate ranking, and recovery of caller-visible contracts. Model outputs will be checked against binary evidence and evaluated against the current rule-based approach using independently annotated reference functions.

6. Conclusions

Copy & Return provides a narrow but useful answer to the safety problem in dynamic ransomware analysis: replace a selected destructive function, apply a controlled redirect, copy, or immediate return, and resume the unchanged caller. The MBRLocker, Conti, and Cylance experiments show that the selected primary damage can be suppressed, while later evidence remains observable. MBRLocker's full sector is captured without writing the actual disk, and the two crypto-ransomware samples rename files, append metadata, and create ransom notes without encrypting the original file bodies.

The analysis extension supports target identification by organizing recovered functions, supporting evidence, and candidate priorities. The 10 selected cases illustrate candidate identification and ranking, while showing that relevant reference functions may appear

outside the first 10 positions. Separately, a compact HGB classifier using numerical static-analysis features achieves a mean F1-score of 0.9124 under nested grouped cross-validation. These results show how static target analysis and AI-based ransomware screening can complement evidence-preserving damage suppression.

The detector classifies binaries, while the analyzer ranks functions for review. Target confirmation and evaluation of the replacement remain separate steps. Future work will improve candidate ranking, test the detector on an independent dataset, and assess whether intervention preserves the operation of benign programs. We will also evaluate the integration of detection, target analysis, and containment.

Author Contributions: Conceptualization, J.B. and S.L.; methodology, J.B., S.-s.L. and S.L.; software, J.B.; validation, J.B., S.-s.L., Y.L. and Y.-j.C.; formal analysis, J.B. and Y.L.; investigation, J.B., S.-s.L. and Y.-j.C.; resources, S.-s.L. and Y.-j.C.; data curation, J.B. and Y.L.; writing—original draft preparation, J.B.; writing—review and editing, all authors; visualization, J.B. and Y.L.; supervision, S.L.; project administration, Y.-j.C. and S.L.; funding acquisition, S.-s.L. and S.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No. RS-2025-02215590, Development of AI implementation obfuscation technology to prevent information leakage in On-Device AI).

Data Availability Statement: The non-executable experimental artifacts and processed records supporting the findings of this study are available from the corresponding author upon reasonable request. Malware binaries cannot be redistributed for security and legal reasons. The evaluated samples were obtained from MalwareBazaar (https://bazaar.abuse.ch, accessed on 25 May 2026) and may be located using the SHA-256 hashes reported in Table 3, subject to the repository's access policies.

Acknowledgments: During the preparation of this manuscript, the authors used OpenAI GPT-6 (accessed on 24 September 2026) for the purposes of editing the text, including rephrasing and related language revision. The authors have reviewed and edited the output and take full responsibility for the content of this publication.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Kharraz, A.; Robertson, W.; Balzarotti, D.; Bilge, L.; Kirda, E. Cutting the Gordian Knot: A Look Under the Hood of Ransomware Attacks. In *Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2015; Volume 9148, pp. 3–24. [CrossRef]
2. Oz, H.; Aris, A.; Levi, A.; Uluagac, A.S. A Survey on Ransomware: Evolution, Taxonomy, and Defense Solutions. *ACM Comput. Surv.* **2022**, *54*, 1–37. [CrossRef]
3. Moser, A.; Kruegel, C.; Kirda, E. Limits of Static Analysis for Malware Detection. In *Proceedings of the Twenty-Third Annual Computer Security Applications Conference*; IEEE: New York, NY, USA, 2007; pp. 421–430. [CrossRef]
4. Egele, M.; Scholte, T.; Kirda, E.; Kruegel, C. A Survey on Automated Dynamic Malware-Analysis Techniques and Tools. *ACM Comput. Surv.* **2012**, *44*, 1–42. [CrossRef]
5. Kharraz, A.; Arshad, S.; Mulliner, C.; Robertson, W.; Kirda, E. UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware. In *Proceedings of the 25th USENIX Security Symposium*; USENIX Association: Berkeley, CA, USA, 2016; pp. 757–772.
6. Kharraz, A.; Kirda, E. Redemption: Real-Time Protection Against Ransomware at End-Hosts. In *Proceedings of the Research in Attacks, Intrusions, and Defenses*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2017; Volume 10453, pp. 98–119. [CrossRef]
7. Bayer, U.; Moser, A.; Kruegel, C.; Kirda, E. Dynamic Analysis of Malicious Code. *J. Comput. Virol.* **2006**, *2*, 67–77. [CrossRef]
8. Roundy, K.A.; Miller, B.P. Hybrid Analysis and Control of Malware. In *Proceedings of the Recent Advances in Intrusion Detection*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6307, pp. 317–338. [CrossRef]
9. Raffetseder, T.; Kruegel, C.; Kirda, E. Detecting System Emulators. In *Proceedings of the Information Security*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4779, pp. 1–18. [CrossRef]

10. Cozzi, E.; Graziano, M.; Fratantonio, Y.; Balzarotti, D. Understanding Linux Malware. In *Proceedings of the 2018 IEEE Symposium on Security and Privacy*; IEEE: New York, NY, USA, 2018; pp. 161–175. [CrossRef]
11. Scaife, N.; Carter, H.; Traynor, P.; Butler, K.R.B. CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data. In *Proceedings of the 2016 IEEE 36th International Conference on Distributed Computing Systems*; IEEE: New York, NY, USA, 2016; pp. 303–312. [CrossRef]
12. Continella, A.; Guagnelli, A.; Zingaro, G.; De Pasquale, G.; Barenghi, A.; Zanero, S.; Maggi, F. ShieldFS: A Self-Healing, Ransomware-Aware Filesystem. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*; ACM: New York, NY, USA, 2016; pp. 336–347. [CrossRef]
13. Kolodenker, E.; Koch, W.; Stringhini, G.; Egele, M. PayBreak: Defense Against Cryptographic Ransomware. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*; ACM: New York, NY, USA, 2017; pp. 599–611. [CrossRef]
14. Elkhail, A.A.; Lachtar, N.; Ibdah, D.; Aslam, R.; Khan, H.; Bacha, A.; Malik, H. Seamlessly Safeguarding Data Against Ransomware Attacks. *IEEE Trans. Dependable Secur. Comput.* **2023**, *20*, 1–16. [CrossRef]
15. Oracle. *Runtime Interposition*. In *Oracle Solaris 11.4 Linkers and Libraries Guide*; Oracle: Austin, TX, USA, 2026. Available online: https://docs.oracle.com/en/operating-systems/solaris/oracle-solaris/11.4/linkers-libraries/runtime-interposition.html (accessed on 27 July 2026).
16. Bernat, A.R.; Miller, B.P. Anywhere, Any-Time Binary Instrumentation. In *Proceedings of the 10th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*; ACM: New York, NY, USA, 2011; pp. 9–16. [CrossRef]
17. Wang, R.; Shoshitaishvili, Y.; Bianchi, A.; Machiry, A.; Grosen, J.; Grosen, P.; Kruegel, C.; Vigna, G. Ramblr: Making Reassembly Great Again. In *Proceedings of the Network and Distributed System Security Symposium*; Internet Society: Reston, VA, USA, 2017. [CrossRef]
18. Wang, Z.; Jiang, X.; Cui, W.; Wang, X.; Grace, M. ReFormat: Automatic Reverse Engineering of Encrypted Messages. In *Proceedings of the Computer Security—ESORICS 2009*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5789, pp. 200–215. [CrossRef]
19. Gröbert, F.; Willems, C.; Holz, T. Automated Identification of Cryptographic Primitives in Binary Programs. In *Proceedings of the Recent Advances in Intrusion Detection*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2011; Volume 6961, pp. 41–60. [CrossRef]
20. Matenaar, F.; Wichmann, A.; Leder, F.; Gerhards-Padilla, E. CIS: The Crypto Intelligence System for Automatic Detection and Localization of Cryptographic Functions in Current Malware. In *Proceedings of the 2012 7th International Conference on Malicious and Unwanted Software*; IEEE: New York, NY, USA, 2012; pp. 46–53. [CrossRef]
21. Lestringant, P.; Guihéry, F.; Fouque, P.A. Automated Identification of Cryptographic Primitives in Binary Code with Data Flow Graph Isomorphism. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*; ACM: New York, NY, USA, 2015; pp. 203–214. [CrossRef]
22. Meijer, C.; Moonsamy, V.; Wetzels, J. Where's Crypto?: Automated Identification and Classification of Proprietary Cryptographic Primitives in Binary Code. In *Proceedings of the 30th USENIX Security Symposium*; USENIX Association: Berkeley, CA, USA, 2021; pp. 555–572.
23. Ren, C.; Zhang, Y.; Xue, H.; Wei, T.; Liu, P. Towards Discovering and Understanding Task Hijacking in Android. In *Proceedings of the 24th USENIX Security Symposium*; USENIX Association: Berkeley, CA, USA, 2015; pp. 945–959.
24. Andronio, N.; Zanero, S.; Maggi, F. HelDroid: Dissecting and Detecting Mobile Ransomware. In *Proceedings of the Research in Attacks, Intrusions, and Defenses*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2015; Volume 9404, pp. 382–404. [CrossRef]
25. Wang, S.; Qin, S.; Qin, J.; Zhang, H.; Tu, T.; Jin, Z.; Guo, J. KRDroid: Ransomware-Oriented Detector for Mobile Devices Based on Behaviors. *Appl. Sci.* **2021**, *11*, 6557. [CrossRef]
26. Shen, Y.; Vervier, P.A.; Stringhini, G. A Large-Scale Temporal Measurement of Android Malicious Apps: Persistence, Migration, and Lessons Learned. In *Proceedings of the 31st USENIX Security Symposium*; USENIX Association: Berkeley, CA, USA, 2022; pp. 1167–1184.
27. Yasui, H.; Inoue, T.; Sasaki, T.; Tanabe, R.; Yoshioka, K.; Matsumoto, T. SPOT: In-Depth Analysis of IoT Ransomware Attacks Using Bare Metal NAS Devices. *J. Inf. Process.* **2024**, *32*, 23–34. [CrossRef]
28. Alsoghyer, S.; Almomani, I. Ransomware Detection System for Android Applications. *Electronics* **2019**, *8*, 868. [CrossRef]
29. Sharma, S.; Krishna, C.R.; Kumar, R. RansomDroid: Forensic Analysis and Detection of Android Ransomware Using Unsupervised Machine Learning Technique. *Forensic Sci. Int. Digit. Investig.* **2021**, *37*, 301168. [CrossRef]
30. Faghihi, F.; Zulkernine, M. RansomCare: Data-Centric Detection and Mitigation against Smartphone Crypto-Ransomware. *Comput. Netw.* **2021**, *191*, 108011. [CrossRef]
31. Xu, R.; Saïdi, H.; Anderson, R. Aurasium: Practical Policy Enforcement for Android Applications. In *Proceedings of the 21st USENIX Security Symposium*; USENIX Association: Berkeley, CA, USA, 2012; pp. 539–552.

32. National Security Agency. Ghidra Software Reverse Engineering Framework. Available online: https://github.com/NationalSecurityAgency/ghidra (accessed on 30 August 2026).
33. angr Project. Control-Flow Graph Recovery (CFG). Available online: https://docs.angr.io/en/latest/analyses/cfg.html (accessed on 31 August 2026).
34. Mandiant. capa: Capabilities in Executable Files. Available online: https://mandiant.github.io/capa/ (accessed on 31 August 2026).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.