iSERVICE: Detecting and Evaluating the Impact of Confused Deputy Problem in AppleOS

Yizhuo Wang, Yikun Hu, Xuangan Xiao, Dawu Gu
Shanghai Jiao Tong University
Shanghai, China
{mr.wang-yz,yikunh,xgxiao,dwgu}@sjtu.edu.cn

ABSTRACT

Confused deputy problem is a specific type of privilege escalation. It happens when a program tricks another more privileged one into misusing its authority. On AppleOS, system services are adopted to perform privileged operations when receiving inter-process communication (IPC) request from a user process. The confused deputy vulnerabilities may result if system services overlook the checking of IPC input. Unfortunately, it is tough to identify such vulnerabilities, which requires to understand the closed-source system services and private frameworks of the complex AppleOS by unraveling the dependencies in binaries.

To this end, we propose iSERVICE, a systematic method to automatically detect and evaluate the impact of confused deputies in AppleOS system services. Instead of looking for insecure IPC clients, it focuses on sensitive operations performed by system services, which might compromise the system if abused, ensuring whether the IPC input is properly checked before the invocation of those operations. Moreover, iSERVICE evaluates the impact of each confused deputy based on i) how severity of the corresponding sensitive operation if abused, and ii) to what extent the sensitive operation could be controlled by external input. iSERVICE is applied to four versions of MacOS (10.14.3, 10.15.7, 11.4, and 12.4) separately. It successfully discovers 11 confused deputies, five of which are zero-day bugs and all of them have been fixed, with three considered high risk. Furthermore, the five zero-day bugs have been confirmed by Apple and assigned with CVE numbers to date.

CCS CONCEPTS

• **Security and privacy → Operating systems security**.

KEYWORDS

AppleOS, confused deputy, privilege escalation, static analysis

ACM Reference Format:
Yizhuo Wang, Yikun Hu, Xuangan Xiao, Dawu Gu. 2022. iSERVICE: Detecting and Evaluating the Impact of Confused Deputy Problem in AppleOS. In *Annual Computer Security Applications Conference (ACSAC '22), December 5–9, 2022, Austin, TX, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3564625.3568001

* Yikun Hu and Dawu Gu are the corresponding authors.

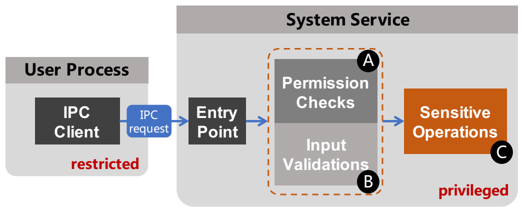


Figure 1: IPC workflow in AppleOS.

1 INTRODUCTION

The confused deputy problem is a security issue where an unprivileged entity could perform actions, which should not have been allowed, by misusing the authority of more privileged ones. AppleOS perform privileged operations with system services that accept IPC request from unprivileged user processes. Despite the strict access control mechanism, attackers could still escalate privileges if system services overlook checking input from IPC requests. However, it is difficult to identify such vulnerabilities in AppleOS, because that requires reverse engineering at a large scale to understand the undocumented system services and Apple's private frameworks on which the services depend.

AppleOS relies on IPC to let restricted user process invoke sensitive operations with privilege. As depicted in Figure 1, an IPC request is proposed by the user process. After passing the protections, i.e., permission checks and input validations, the data carried by the request (i.e., IPC input) is executed as parameters of the sensitive operations. If system services lack such checks or check the IPC request in an improper way, the confused deputy would result.

IPC-based confused deputy vulnerabilities are not new. Unfortunately, existing work cannot tackle the problems at its root. On the one hand, existing work for confused deputy detection only focuses on permission checks without diving into the input validations. Kobold [16] sets its sights on entitlements in third-party iOS applications, which are used for permission checks, while it disregards the contents of IPC request, i.e., how the input is validated. Similarly, work on Android [1, 9, 19, 20, 22, 23, 31] mainly studies mis-configured issues of permission checks of system services. On the other hand, fuzzing-based methods [13, 26] rely on crashes caused by program inputs, while confused deputies are not memory bugs but privilege escalation, which do not produce crashes.

In this paper, we propose iSERVICE, a systematic method to detect and evaluate the impact of confused deputies on AppleOS in a static manner. Essentially, confused deputies become harmful if they are tricked into invoking sensitive operations with privilege. Therefore, iService concentrates on those operations, checking whether the IPC input reaching sensitive operations has been well validated. iService takes four steps to achieve the goal: i) resolve function calls using top-down type propagation, thereby generating the call graph, ii) identify sensitive operations on the call graph, iii) perform sensitive operation-oriented dataflow analysis to identify data dependencies between IPC inputs and sensitive operations, thereby extracting input validations without fixed pattern, iv) evaluating input validations to report confused deputies.

iService is applied to 439 system services of macOS 10.14.3, 10.15.7, 11.4 and 12.4, and discovered 28 services where the key parameters of 53 sensitive operations are controllable by IPC inputs. By extracting and evaluating the protections of these controllable sensitive operations, iService reports 20 confused deputies due to lacking proper permission checks or input validations. We manually identified 11 of them that could be exploited, including arbitrary file overwriting and commands execution. We have reported all of them to Apple, and 5 of them are 0-day vulnerabilities, which are confirmed with CVE numbers assigned.

In summary, this paper makes the following contributions:

- We present iService, the first systematic method to detect and evaluate the impact of confused deputies on AppleOS.
- We perform top-down type propagation to statically resolve function calls which are dynamically dispatched, thereby identifying sensitive operations in system services.
- We propose a sensitive operation-oriented dataflow analysis to identify data dependencies between IPC inputs and sensitive operations, thereby extracting input validations with no fixed patterns.
- We implement a prototype of iService and evaluate it with 439 system services in four AppleOS. iService discovered 11 confused deputies without proper permission checks or input validations, which lead to privilege escalation. 5 of them are confirmed with the CVE numbers assigned.

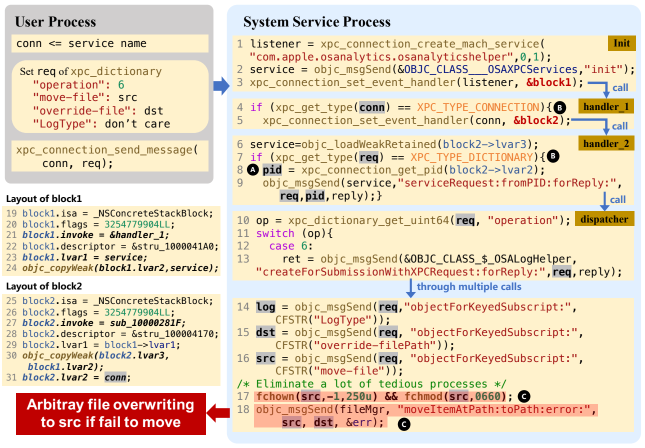
2 MOTIVATION AND OVERVIEW

In this section, we first unravel the limitations of existing work and corresponding challenges with a motivating example. Then, we explain the basic idea of iService and the overview of its design.

2.1 Motivating Example and Challenges

Figure 2 provides an example of a confused deputy vulnerability, which allows a malicious program to overwrite arbitrary files for root privilege. The system service in Figure 2 accepts a dictionary `req` as input and finally performs two operations in sequence, i.e., manipulating file permissions (Line 17) and moving files (Line 18). The two operations are considered to be ***sensitive operations*** *because they are functions requiring elevated privileges to execute* [2].

Since there is no validation to restrict the arguments used in the two sensitive operations, the service can be abused to manipulate file permissions and overwrite files. Specifically, if the target file already exists, the service fails to move the file, but it changes the permission of the source file. Then, a confused deputy vulnerability results because there is a lack of checks for the service to validate whether `dst` (Line 15) has existed before moving



```
conn <= service name

Set req of xpc_dictionary
  "operation": 6
  "move-file": src
  "override-file": dst
  "LogType": don't care

xpc_connection_send_message(
    conn, req);
```

```
1 listener = xpc_connection_create_mach_service(
  "com.apple.osanalytics.osanalyticshelper",0,1);
2 service = objc_msgSend(&OBJC_CLASS___OSAXPCServices,"init");
3 xpc_connection_set_event_handler(listener, &block1);

4 if (xpc_get_type(conn) == XPC_TYPE_CONNECTION){
5   xpc_connection_set_event_handler(conn, &block2);

6 service=objc_loadWeakRetained(block2->lvar3);
7 if (xpc_get_type(req) == XPC_TYPE_DICTIONARY){
8   pid = xpc_connection_get_pid(block2->lvar2);
9   objc_msgSend(service,"serviceRequest:fromPID:forReply:",
      req,pid,reply);}

10 op = xpc_dictionary_get_uint64(req, "operation");
11 switch (op){
12   case 6:
13     ret = objc_msgSend(&OBJC_CLASS_$_OSALogHelper,
   "createForSubmissionWithXPCRequest:forReply:",req,reply);

14 log = objc_msgSend(req,"objectForKeyedSubscript:",
      CFSTR("LogType"));
15 dst = objc_msgSend(req, "objectForKeyedSubscript:",
      CFSTR("override-filePath"));
16 src = objc_msgSend(req, "objectForKeyedSubscript:",
      CFSTR("move-file"));
/* Eliminate a lot of tedious processes */
17 fchown(src,-1,250u) && fchmod(src,0660);
18 objc_msgSend(fileMgr, "moveItemAtPath:toPath:error:",
      src, dst, &err);
```

```
19 block1.isa = _NSConcreteStackBlock;
20 block1.flags = 3254779904LL;
21 block1.invoke = &handler_1;
22 block1.descriptor = &stru_1000041A0;
23 block1.lvar1 = service;
24 objc_copyWeak(block1.lvar2,service);
```

```
25 block2.isa = _NSConcreteStackBlock;
26 block2.flags = 3254779904LL;
27 block2.invoke = sub_10000281F;
28 block2.descriptor = &stru_100004170;
29 block2.lvar1 = block1->lvar1;
30 objc_copyWeak(block2.lvar3,
    block1.lvar2);
31 block2.lvar2 = conn;
```

Figure 2: CVE-2021-30774, a confused deputy which allows a malicious program to gain root privilege.

`src` (Line 16). Hence, to detect such vulnerability, the main task is to determine whether the key parameters of sensitive operations are well-validated. That faces two challenges: i) resolving Objective-C messages, and ii) identifying sensitive input validations.

2.1.1 ***Challenge 1: Resolving Objective-C Messages.*** Most of the sensitive operations in this paper are function invocations in Objective-C, performed via an indirect call, namely message passing. It is implemented by `objc_msgSend`, a dynamic dispatch function, such as Line 18 in Figure 2. Therefore, to identify sensitive operations, it is necessary to find the proper function to invoke, i.e., resolving message passing.

Nevertheless, resolving such a dynamic process is difficult for static methods that require to find the type of the object to which the message is passed. Existing work performs backward slicing [15, 18], which cannot find the type propagated interprocedurally. For example, the type of `service`, which is the first parameter of `objc_msgSend` (Line 9), is defined in another function (Line 2) and propagated across functions via structure fields `block1.lvar2` and `block2.lvar3`. It is hard for backward slicing to figure out its type, thus cannot resolve the message in Line 9.

2.1.2 ***Challenge 2: Identifying Sensitive Input Validations.*** On the one hand, Input validations have no fixed pattern like permission checks and vary for different inputs. On the other hand, a lack of validations for inputs on which sensitive operations depend, namely sensitive input validations, could lead to confused deputies. Therefore, to precisely detect confused deputies, it is necessary to find all sensitive input validations without over-approximation.

Existing work focuses on conditional comparison or error handling instructions. However, it does not consider whether sensitive operations depend on these instructions, which causes false positives [1, 9, 21, 33]. For example, existing work considers a lack of restrictions for `log` (Line 14), a field of IPC input `req`, thereby reporting a confused deputy. It is a false positive because `log` is irrelevant to any sensitive operation.

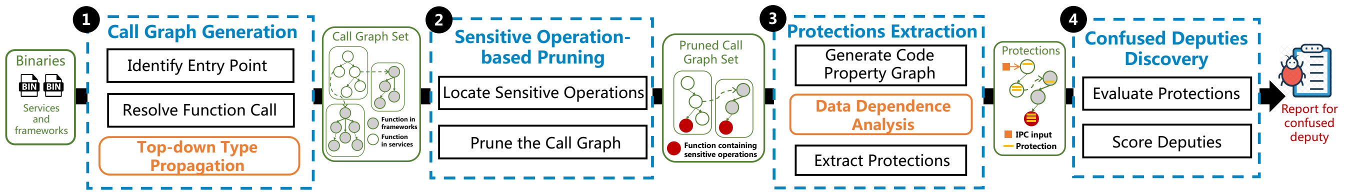


Figure 3: Overview of iService

2.2 Overview

To this end, we design iService, a static analysis framework to detect confused deputies. Figure 3 depicts the workflow of iService, which consists of four steps:

(1) **Call Graph Generation.** iService first identifies the starting function (`handler_1` in Figure 2), performs top-down type propagation to resolve Objective-C messages (*Challenge 1*), and constructs a call graph.
(2) **Sensitive Operation-based Pruning.** iService then identifies sensitive operations (Lines 17, 18) on the call graph and prunes the call graph to filter out functions irrelevant to sensitive operations, such as other functions called by function `dispatcher` in the `switch-case` code block in Figure 2.
(3) **Protections Extraction.** iService generates the code property graph [32] for functions in the pruned call graph, and performs dataflow analysis to find data dependencies between the IPC input and key parameters of sensitive operations, such as *req* ⟶ *src* (Line 18 in Figure 2). Based on the data dependencies, iService extracts protections, i.e., permission checks and sensitive input validations (*Challenge 2*).
(4) **Confused Deputies Discovery.** iService evalutes protections extracted, and reports confused deputies based on a scoring mechanism, which indicates the possibility of the exploitability. For example, The service in Figure 2 scores 2 [1] since it has no validation for `src` and `dst` used in sensitive operations. Thus iService reports it as a confused deputy.

2.2.1 *Top-down Type Propagation.*

To handle Challenge 1, we propose the top-down type propagation, thereby inferring the type of object receiving the messages. The insight is that the object receiving messages is commonly defined via class instantiate function or return values of standard library functions, which introduce the type information. Therefore, iService performs type propagation in a top-down manner. Specifically, it introduces and propagates types along the control flow of each function to infer types needed in the call site. After resolving a function call, iService passes the type information in the caller's context to the callee for further propagation in the callee's context.

2.2.2 *Sensitive Operation-oriented Data Dependence Analysis.*

To solve Challenge 2, we propose a data dependence analysis which is sensitive operation-oriented. The insight is that sensitive input validations should restrict the value of input fields on which the key parameters of sensitive operations depend. Therefore, iService finds data dependencies between IPC inputs and key parameters of sensitive operations via dataflow analysis. Based on the data dependencies, iService identifies statements that restrict the dataflow between inputs and key parameters as sensitive input validations without the requirement to specify a fixed pattern.

[1]A lower score means weaker protection and therefore more likely to be exploited.

3 DESIGN

In this section, we explains the four steps of iService in details. By performing type propagation, iService resolves function calls and generates the call graph of the system service (§ 3.1). It then locates sensitive operations on the call graph and pruned the call graph based on them (§ 3.2). To extract protections on the pruned call graph, i.e., permission checks and sensitive input validations, iService performs data dependence analysis (§ 3.3). Finally, it evaluates protections and reports confused deputies (§ 3.4).

3.1 Call Graph Generation

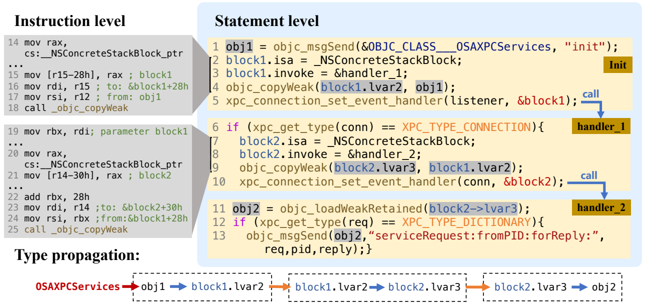
iService recovers call relationships and generates the call graph of the service binary in a top-down manner. It first identifies the IPC entry point of the binary, which is the function to accept and handle the IPC request (§ 3.1.1). iService then resolves function calls in the entry point and its callees iteratively, thereby generating the call graph starting from the entry point. Specifically, iService performs top-down type propagation to resolve functions invoked via Objective-C messages (§ 3.1.2).

3.1.1 *IPC Entry Points Identification.*

IPC entry points are the interface functions that an Apple system service exposes to user-space programs, which handle IPC requests and are the essential starting points of system service analysis and bug detection. iService identifies entry points of the most used IPC mechanism in AppleOS, called XPC, which has two types of implementations, i.e., C-based XPC and Objective-C-based NSXPC.

C-based XPC entry points are set by the function called `xpc_connection_set_event_handler`. This function accepts two parameters, the first is the listener of the service, and the second is a structure, namely `StackBlock`. The structure has a function pointer field that points to the IPC entry point. Take Figure 2 as an example, iService first finds the listener of the service via `create` related APIs in Line 1. It then slices forward to find the function used to set the event handler in Line 3. By recovering the function pointer field in Line 21, iService identify the entry point which is labeled as `handler_1` in Lines 4, 5.

NSXPC entry points are exported methods of the delegate class of the remote procedure call mechanism. Therefore, iService finds the delegate class and dumps its exported methods as entry points. In details, iService first identifies the `NSXPCListener` instance, to which the service name and delegate instance bind. It then analyzes the method `listener:shouldAcceptNewConnection:` of the delegate instance, which is used to authenticate the IPC client and set the connection. iService identifies exported object set in the



```
14 mov rax,
   cs:__NSConcreteStackBlock_ptr
...
15 mov [r15-28h], rax ; block1
16 mov rdi, r15 ; to: &block1+28h
17 mov rsi, r12 ; from: obj1
18 call _objc_copyWeak

19 mov rbx, rdi; parameter block1
...
20 mov rax,
   cs:__NSConcreteStackBlock_ptr
21 mov [r14-30h], rax ; block2
...
22 add rbx, 28h
23 mov rdi, r14 ;to: &block2+30h
24 mov rsi, rbx ;from:&block1+28h
25 call _objc_copyWeak
```

```
1 obj1 = objc_msgSend(&OBJC_CLASS___OSAXPCServices, "init");
2 block1.isa = _NSConcreteStackBlock;
3 block1.invoke = &handler_1;
4 objc_copyWeak(block1.lvar2, obj1);
5 xpc_connection_set_event_handler(listener, &block1);

6 if (xpc_get_type(conn) == XPC_TYPE_CONNECTION){
7   block2.isa = _NSConcreteStackBlock;
8   block2.invoke = &handler_2;
9   objc_copyWeak(block2.lvar3, block1.lvar2);
10  xpc_connection_set_event_handler(conn, &block2);

11 obj2 = objc_loadWeakRetained(block2->lvar3);
12 if (xpc_get_type(req) == XPC_TYPE_DICTIONARY){
13   objc_msgSend(obj2,“serviceRequest:fromPID:forReply:”,
        req,pid,reply);}
```

Figure 4: An example to reveal the details of the type propagation. iService lifts the binary from instruction level (left) to statement level (right). The statements in the figure are simplified for better understand.

method, and dumps its methods as the entry points of the service by scanning corresponding sections of the binary.

3.1.2 *Top-down Type Propagation.*

iService performs intra- and inter-procedural type propagation in a top-down manner, and resolves function calls based on the types inferred.

Intra-procedural type propagation. iService traverses each functions along the control flow, and propagates type information by maintaining a object-type map recording inferred types. It lifts the binary from the instruction level to the statement level, where objects is referenced by registers and memory address, as is shown in Figure 4. During the traversal, iService updates the map by introducing or propagating type information based on the statements' operators and operands. Specifically, iService summarizes three sources to introduce type information:

- *Function parameters.* When a statement uses a function parameter, the corresponding type is introduced.
- *Function prototypes.* Standard library and instantiation functions have standard prototypes, which introduce types of the return value and arguments.
- *Recovered structures.* Structures, such as StackBlock used to invoke anonymous functions, introduce types of their fields when corresponding fields are used.

The types are propagated via the following two ways:

- *Assignment-like statements.* iService assigns the type of the source operand to the destination operand.
- *Prototypes of private functions resolved.* iService generates prototypes of resolved private functions and infers types of variables involved in other call sites.

Inter-procedural type propagation. iService performs type propagation across functions in a top-down manner while building the call graph. It first finishes intra-procedural type propagation before the call site and then propagtes types into the callee. Specifically, iService propagates type information across functions via i) arguments of function calls, and ii) StackBlock which is the structure used to invoke an anonymous function.

Arguments passed at the call site propagate types into the callee's context. iService records arguments' types at the call site into the callee's object-type map. The recorded type is introduced when a statement uses an argument in the callee's context.

As for StackBlock, iService recovers its layout and records the types of its fields to the object-type map of the anonymous function invoked by it. Its layout contains four fields of intrinsic properties, and other fields store local variables as member variables of StackBlock. The definition and usage of StackBlock are one-time, which means the assignment-like statements between them build the layout. Therefore, iService identifies the statements that declare and use StackBlock respectively, and labels the stack variables assigned between the two statements as member variables. After that, iService records the types of these member variables. When these member variables are used in the anonymous function's context, iService introduces corresponding types.

Example. Take the type inferring of obj2 in Figure 4 as an example. The instantiation function in Line 1 introduces the type called OSAXPCServices and assigns it to the return value obj1. iService identifies assignment-like statements (Lines 2-4), which build the layout of block1, a StackBlock passed into the function handler_1. The statement in Line 4 copies obj1 to block1.lvar2. Therefore, iService infers that the type of block1.lvar2 is OSAXPCServices as well. After that, iService finds that the type of block2.lvar3 is the same as block1.lvar2 in Line 9. Similarly, iService propagates the type of block2.lvar3 into function handler_2 and inferred that the type of obj2 is OSAXPCServices (Line 11). Knowing the type of object receiving the message (i.e., obj2), iService resolves the function call in Line 13.

3.2 Sensitive Operation-based pruning

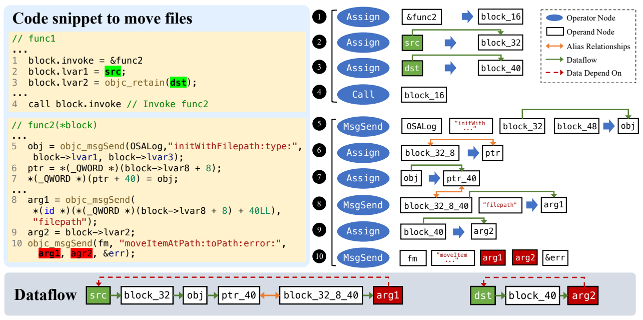
iService identifies sensitive operations and prunes the call graph based on them. It summarizes four categories of sensitive operations based on Apple's secure coding guide [2], as follows:

- **File Permissions Manipulation.** These operations may be abused to downgrade the access control of system files for further tempering, such as fchown and fchmod.
- **System and User Files Manipulation.** These operations directly access files with the root privilege of system services, which may be abused for arbitrary file overwriting, such as reading, writing, moving, and deleting files.
- **Preferences Management.** User and application preferences are stored as key-value pairs. Operations, which are used to get, set, synchronize, add and remove preferences, may be abused for preference leakage or tampering via manipulated keys.
- **Processes Management.** These operations can be abused to execute specific files or commands with certain parameters for code execution with root privileges, such as operations on NSTask which represents a subprocess.

According to the four categories above, iService collects a total of 40 C-APIs and 94 Objective-C APIs from Apple's public documents, as listed in Appendix A. It also identifies the key parameters of them, which can lead to privileges escalation if being controlled.

iService then prunes the call graph only to keep the functions that directly or indirectly affect the execution of sensitive operations for further analysis.

- *Direct Effect.* The functions in the function calling path directly affect sensitive operations. For example, function f_k,


Code snippet to move files

```
// func1
...
1  block.invoke = &func2
2  block.lvar1 = src;
3  block.lvar2 = objc_retain(dst);
...
4  call block.invoke // Invoke func2

// func2(*block)
...
5  obj = objc_msgSend(OSALog,"initWithFilepath:type:",
    block->lvar1, block->lvar3);
6  ptr = *(_QWORD *)(block->lvar8 + 8);
7  *(_QWORD *)(ptr + 40) = obj;
...
8  arg1 = objc_msgSend(
    *(id *)(*(_QWORD *)(block->lvar8 + 8) + 40LL),
    "filepath");
9  arg2 = block->lvar2;
10 objc_msgSend(fm, "moveItemAtPath:toPath:error:",
     arg1, agr2, &err);
```

Figure 5: An example for data dependence analysis. Code in the left is lifted from binary for better understand. Statements in the right are parts of the code property graph. iService constructs dataflow on the CPG, and extracts data dependencies between input fields (e.g., `src`) and key parameters (e.g., `arg1`) of the sensitive operation in Line 10.

called via $f_{entry} \rightarrow f_i \rightarrow f_j \rightarrow f_k$, invokes the sensitive operation, then f_{entry}, f_i, and f_j are considered directly affect the sensitive operation.

- *Indirect Effect.* The functions called before the call sites of the function calling path indirectly affects sensitive operations. For example, in the body of function f_j, if f_m is called before the call site of f_k, then the f_m is considered indirectly affect the sensitive operations invoked in f_k.

3.3 Protections Extraction

To extract protections, especially sensitive input validations without fixed patterns, iService identifies data dependencies between IPC inputs and key parameters of sensitive operations based on the insight that sensitive input validations should restrict the dataflow between between them. Specifically, iService first generates *code property graph* (CPG) [32] of functions in the pruned call graph(§ 3.3.1), and then constructs interprocedural dataflow on the CPG(§ 3.3.2), which is transformed into a graph-reachability problem [28, 29]. After that, it identifies protections of sensitive operations based on the data dependencies(§ 3.3.3).

3.3.1 *Generate Code Property Graph.*

The code property graph is a joint data structure consisting of three levels of information, i.e., abstract syntax, control flow, and both data and control dependencies. Unlike the abstract syntax tree decomposed from source code, the abstract syntax here refers to statements lifted from the binary, and each statement contains an operator and its operands, which are not nested. By adding control flow and both data and control dependencies to the statements, iService generates the code property graph to represent each function.

Specifically, a code property graph $G = (V, E, \lambda, \mu)$ is a directed, edge-labeled, attributed multigraph, where V is a set of nodes representing operators and operands of abstract syntax, E is a set of directed edges representing relationships among operators and operands, control flow and dependencies among statements. Labeling function λ assigns labels to different types of edges, while μ assigns property value to each node according to the operator or operand the node represents.

iService abstracts the following five operators as nodes to lift the binary:

- *Assignment.* It assigns from a source operand to a destination operand and introduces data dependencies.
- *Arithmetic.* It operates on source operands and stores the result in the target operand.
- *Call.* It represents C-based functions that use several operands as arguments and returns an operand as the return value.
- *MsgSend.* It represents the Objective-C messages that use several operands as its receiver, selector, and parameters respectively, and returns an operand as the return value.
- *Jump.* It represents conditional instructions that introduce control dependencies.

The operands of *Assignment* contain both registers and memory objects. The former are identified using register name and *Assignment*'s location in the CPG. The latter are identified using the base address and offset value computed in *Arithmetic*. For example, `block_32` in Figure 5 is the field at offset 32 in the structure `block`. In addition, for ARC-related methods added by the compiler, such as `objc_retain`, `objc_copyweak`, etc., iService also abstracts them as *Assignment* because of their semantics, such as Statement 3 in Figure 5.

3.3.2 *Construct Interprocedural Dataflow Based on CPG.*

To identify data dependencies between inputs and key parameters of sensitive operations, iService constructs interprocedural dataflow based on CPG in a bottom-up manner. It traverses functions of the pruned call graph in post-order, which ensures a function will only be analyzed after all its callees are analyzed. For each function analyzed, iService generates a *summary of data dependencies* among parameters and the return value, which represents the effects among them. Specifically, iService constructs dataflow among operands according to the types of operator:

- *Assignment* and *Arithmetic.* iService connects the dataflow directly between source and destination operands.
- *MsgSend* and *Call.* iService connects the dataflow among parameters and the return value according to the callee's *summary of data dependencies* generated before.

Using such a summary-based method, iService constructs interprocedural dataflow without repeatedly and iteratively analyzing each function invoked. As for the ring-shaped call relationship in the callgraph, iService considers that the return value is affected by all its parameters for soundness.

To precisely capture the dataflow of fields of structured IPC input, iService performs alias analysis to fulfill field-sensitive dataflow. It identifies memory objects using the base address and offset value, which are used as operands in *Arithmetic*, and the base address are nested when used in multi-level pointers. From the *Assignment* using memory objects as operands, iService records alias relationships between source and destination operands. And the fields of two operands with alias relationships are also labeled as alias if they have the same offset.

Example. Take the data dependence analysis for sensitive operation (Line 10) in Figure 5 as an example. According to the bottom-up manner, iService first generates summary of data dependencies of functions invoked in Lines 5, 8, 10. After that, iService connects dataflow among operands of corresponding *MsgSend* statements, such as return value `obj` and parameter `block_32` of Statement 5. As for alias analysis, iService first identifies alias relationship between pointers `block_32_8` and `ptr` in Statement 6, and then infers the alias relationship between pointers `block_32_8_40` and `ptr_40` because of the alias base and the same offset. Finally, iService constructs the dataflow from `src` to `arg1` and extracts the data dependencies between them, which means the sensitive operation in Line 10 is dependent on inputs.

3.3.3 ***Extract Protections.*** To extract protections of sensitive operations, especially sensitive input validations without fixed patterns, iService uses the extracted data dependencies and dataflow on the CPG. The insight is that sensitive input validations are used to restrict the parameters of sensitive operations dependent on inputs. Therefore, iService traverses the CPG along each dataflow from IPC inputs to parameters of sensitive operations. For each *Jump* met, iService checks the data dependencies of its conditions. If a condition has data dependencies with both IPC inputs and key parameters of sensitive operations, iService identifies the *Jump* as sensitive input validation.

3.4 Confused Deputy Discovery

To decided whether sensitive operations are well-protected, iService evaluates permission checks (§ 3.4.1) and sensitive input validations (§ 3.4.2) extracted above. iService further computes a score to report a confused deputy and indicate the possibility of exploitability (§ 3.4.3).

3.4.1 ***Evaluate Permission Checks.*** iService determines lacking permission checks according to its inspection target and method [8, 24, 30]. The permission check validates the identity of the IPC client via examing its executable file. For example, the key-value pair called entitlement grants executable permission to use a service, which is statically embedded into the binary. Specifically, iService sets the following four levels for permission checks according to the object they check:

(1) *No permission check.* The system service directly accepts the connection without authenticating the IPC client.
(2) *Weak permission check.* The system service only checks bundle ID or static code, etc.
(3) *PID-based permission check.* The system service locates the process via PID and then checks the entitlement.
(4) *Audit token-based signature check.* The system service locates the process via audit token to check the entitlement.

The four levels ascend from low to high security, where only permission checks of Level 4 are sufficient. Validations of Level 3 are also risky due to locating the process via PIDs. PIDs could be reused via a race to trick code signature verification into checking different binaries due to the relatively small PID space of the OS, whereas locating the process with an audit token does not [24].

3.4.2 ***Evaluate Sensitive Input Validations.*** iService detects a lack of input validations based on the insight that input valildations should restrict the value range of the key parameter of sensitive operations. The key parameter could be used to exploit the sensitive operation and is marked during collecting sensitive operations from Apple's public documents in Section 3.2. If there is no input validation for the key parameter, iService reports a missing check. If input validations exist, iService further evaluates the sufficiency of input validations.

According to the degree of restriction on the value range, iService divides sensitive input validations into four levels:

(1) *No sensitive input validation.* The system service does not validate sensitive inputs or only check if they are non-null.
(2) *Universal validation.* The validation only checks the type or length and do not restrict the value of a specific parameter.
(3) *Weak validation specific to the parameter.* The validation restricts the sensitive parameter to a certain value range.
(4) *Strong validation specific to the parameter.* The validation restricts the sensitive parameter to a certain value.

Four levels of security ascend from low to high, and iService considers a lack of input validations if no value range or only a single-side value range is restricted. Taking the moving file operation as an example, both prefix check (Level 3) and equivalent check (Level 4) validates the file path parameter via the sting API. The prefix check can be bypassed using path traversal, while the equivalent check (e.g., `strcmp`) cannot.

3.4.3 ***Score Confused Deputies Based on Level Weight.*** iService provides a scoring mechanism to indicate the possibility of the exploitability of confused deputies. It scores a deputy from two dimensions, i.e., permission checks and input validations. The permission check is performed once for an IPC connection and its evaluation is to determine the level according to Section 3.4.1.

As for the input validations, cases are much more complicated. For a sensitive operation, each parameter from IPC input may have multiple dataflows and the input validations of each dataflows may be different.

Therefore, iService performs a level weight scoring mechanism to evaluate input validations according to Section 3.4.1. Precisely, it computes a value to represent the possibility of exploitability for each sensitive operation, denoted as E, which is defined as follows:

$$E = \frac{1}{n} \sum_{Dset_i} \min_{D_j \ in \ Dset_i} \Big(\sum_{IV_k \ in \ P_j} m_k \, W_k \, IV_k \Big) \tag{1}$$

where n represents the number of sensitive operation's parameters controlled by the input, $Dset_i$ represents the set of dataflows from the IPC input to i^{th} parameter, D_j represents one of the dataflows in $Dset_i$. Inside, IV_k and W_k represent the k^{th} sensitive input validation of the dataflow and its level weight, respectively, and m is a bool value that indicates whether IV_k is repeated.

The formula uses sensitive input validations on the most exploitable execution path to characterize the exploitability of the controllable parameters of the sensitive operation. It then averages the exploitability of controllable parameters to measure the exploitablity of the sensitive operation. The lower the score, the more possible it is to be exploited.

According to the mechanism above, iService computes scores for sensitive operations in system services and report services with E below pre-defined threshold and permission check level less than four as confused deputies. The score can also help the analyzer to learn the severity of the vulnerability.

4 IMPLEMENTATION

iService is written in Python totaling about 4,300 LOC. It uses IDA Pro [25] to disassemble the Mach-O binary and lifts the code to Microcode, the intermediate language of IDA. Both the type propagation (§ 3.1.2) and code property graph generation (§ 3.3.1) are built on top of Microcode. The data dependence analysis (§ 3.3.2) is implemented on the graph database Neo4j [27]. Specifically, iService stores the code property graph in the Neo4j database and constructs interprocedural dataflow on the graph via Neo4j. In addition, both point-to and alias relationships are processed as edges among nodes representing operands on the graph. By querying on the graph database, iService extracts data dependencies between operands as paths on the graph and identifies protections from statements along the data dependencies.

As for the threshold of the scoring mechanism, iService uses an empirical value calculated according to Formula 1. Since a key parameter of the sensitive operation is considered well-validated by at least one input validation of Level 4, the vector IV is set to be $[0, 0, 0, 1]$, and the vector W is set to be $[1, 2, 3, 4]$ as default. Therefore, the threshold value is computed to be four. Any score lower than four is considered to be exploitable.

5 EVALUATION

In this section, we evaluate iService with 1,256 daemon binaries from four versions of AppleOS to show its effectiveness and performance. We first assess the effectiveness, including the overall results and confused deputy findings (§ 5.2). Then we evaluate individual analysis modules (§ 5.3 and § 5.4) and the performance of iService (§ 5.5). At last, we use two CVEs found by iService as a case study to demonstrate its effectiveness further(§ 5.6).

5.1 Setup

5.1.1 ***Hardware.*** Our experiment is conducted on MacBook Pro 2019 with macOS 10.15.7, Intel(R) Core(TM) i9-9980HK CPU @ 2.40GHz and 64 GB RAM.

5.1.2 ***Dataset.*** We collected a total of 1,256 daemon binaries from four versions of AppleOS, including macOS 10.14.3, 10.15.7, 11.4, and 12.4. Since the closed-source AppleOS does not document system services and their paths, we identified system services from daemons. We got the daemons through `launchd`, the first user-mode process on boot and the parent of all userspace processes. In addition to system services, many system programs also run as daemons but are not accessible for user processes. Therefore, we extracted system services by scanning whether there are services created via XPC or NSXPC in these binaries. Finally, we extracted 439 binaries registered as system services through XPC or NSXPC from 1,256 daemon binaries as experiment objects.

5.1.3 ***Metrics.*** Similar to other work [15, 18], we adopt Precision as the metric for evaluation, as defined as $Precision = \frac{tp}{tp+fp}$, where tp means the bug reported is exploitable, fp means the bug reported is unexploitable, and fn means the exploitable bug is not reported. Due to the lack of ground truth of vulnerabilities in closed-source AppleOS, it is impractical to report Recall.

5.2 Effectiveness

5.2.1 ***Overall Results.*** We applied iService to the dataset and completed the experiment in 16 hours. Specifically, iService identified 439 system services that can be invoked via XPC or NSXPC in 1,256 daemon binaries and located 830 entry points of them, as shown in Table 1. Furthermore, iService found 431 sensitive operations and determined that 87 of them depended on the input via data dependence analysis. After that, iService extracted and evaluated permission checks and input validations based on the data dependencies. Finally, iService reported 20 confused deputies, of which 11 were confirmed to be exploitable.

5.2.2 ***Bug Findings.*** Altogether, iService reported a total of 20 confused deputies in 439 service binaries, 4 of which were duplications in multiple versions. We manually identified 11 unique true positives that could be exploited, including overwriting system files, executing commands, etc., as listed in Table 2. We developed PoC for all 11 confused deputies and reported them to Apple. 5 of them were 0-days vulnerabilities and assigned CVEs with Apple acknowledgement, i.e., CVE-2021-30774 [6], CVE-2019-8521, CVE-2019-8565, CVE-2019-8513, and CVE-2019-8530[3] [2]. The other 6 were 1-days vulnerabilities. At present, these vulnerabilities have been fixed by Apple.

To demonstrate the severe impact of these vulnerabilities, we developed proof-of-concept user-space programs to exploit them in the corresponding OSes. As shown in Table 2, We found that 2 of them allow malicious user programs to gain root privilege, 4 of them allow arbitrary file writing, 4 of them may cause DoS, and 1 of them allow arbitrary shell command execution. In addition, these vulnerabilities also affect iOS and tvOS [4, 5, 7].

5.2.3 ***Precision.*** Among 20 confused deputies reported, 11 are true positives, of which 5 are assigned with CVE numbers. 9 are false positives, of which 4 are duplications in multiple versions. Therefore, the precision is 55%. Since the total amount of reports is not large, the precision is feasible for manual confirmation of vulnerability exploitability.

False positives are caused by failure to locate functions in private system libraries. After macOS BigSur, all system libraries are combined into one large cache file called dyld shared cache, resulting in iService being unable to dump the hierarchy of classes and locate their methods. Specifically, the false positives do not use public APIs to check permission and validate inputs but use methods in private libraries. The two private methods they used, i.e., `DEUtilsValidateConnection` and `DEUtilsValidateDestinition`, correctly check the IPC client's entitlement and validate the input. Since failing to locate these two functions in the dyld shared cache, iService incorrectly determined that these services missed permission check and input validation, resulting in false positives.

False negatives are caused by two reasons. On the one hand, some services use IPC mechanisms implemented in the private library, making iService unable to identify their entry points for

[2]The four CVEs in 2019 are disclosed by our industry collaborator Zhi Zhou (CodeColorist) by adopting iService.

Table 1: Result of the sensitive operation identification

AppleOS	Daemon	System Service			Entry Points			Sensitive Operation							
		XPC	NSXPC	Total	XPC	NSXPC	Total	A	B	C	D [1]	In service	In Framework	Total	Controllable[2]
macOS 10.14.3	293	82	16	98	80	25 (76)	105 (156)	7	38	11	58	105	11	114	31
macOS 10.15.7	301	88	19	107	88	27 (72)	115 (160)	10	39	10	59	106	12	118	27
macOS 11.4	325	96	24	120	113	64 (166)	177 (279)	3	40	11	53	117	–	117	17
macOS 12.4	337	99	15	114	113	69 (122)	182 (235)	2	32	14	21	82	–	82	12
Total	1256	365	74	439	394	185 (436)	579 (830)	29	127	37	194	368	23	431	87

1: A refers to sensitive operations of *File Permissions Manipulation.* B refers to sensitive operations of *System and User Files Manipulation.* C refers to sensitive operations of *Preferences Management.* D refers to sensitive operations of *Process Management.* 2: Controllable means a sensitive operation has at least one parameter depending on IPC inputs.

Table 2: Summary of confused deputy vulnerabilities found.

Vulnerable Service	OS	Cause	Impact	Status
osanalyticshelper	macOS 10.15, 11	Missing permission checks and input validations	Gain root privilege	CVE-2021-30774
fbahelperd	macOS 10.12-14	Weak permission checks and input validations	Overwrite arbitrary files	CVE-2019-8521
fbahelperd	macOS 10.12-14	Weak permission checks and input validations	Gain root privilege	CVE-2019-8565
timemachinehelper	macOS 10.12-14	Missing permission checks and input validations	Execute arbitrary shell command	CVE-2019-8513
timemachinehelper	macOS 10.12-14	Missing permission checks and input validations	Overwrite arbitrary files	CVE-2019-8530
bluetoothhelper	macOS 10.12-14	Missing permission checks and input validations	Overwrite arbitrary files	Repaired
getmobilityinfohelper	macOS 10.12-14	Missing permission checks and input validations	Overwrite arbitrary files	Repaired
fud	macOS 10.14-15	Weak permission checks and input validations	File access and DoS	Repaired
storelegacy	macOS 10.14-15	Missing permission checks and input validations	File access and DoS	Repaired
coresymbolicationd	macOS 10.15, 11	Missing permission checks and input validations	May lead to DoS	Repaired
wifihelper	macOS 10.12-14	Missing permission checks and input validations	File access and DoS	Repaired

further analysis. For example, service `backupd-helper` uses a private library to implement a specific XPC called `TMXPC`, thus iService failed to analyze it. On the other hand, iService suffers from inherent issues of dataflow analysis in binary. For example, the over-approximate alias set will cause a state explosion. In practice, iService cannot handle all those states, which results in false negatives of controllable sensitive opertions. In addition, IDA would make mistakes or even fail when lifting complex functions to its IR for reasons like stack pointer recognition error, indirect call recovery error, etc., which results in failure to analyze these services.

5.3 Sensitive Operation Identification.

5.3.1 ***Entry Point Identification.*** iService identified 830 entry points from 439 system services. As explained in Section 3.1.1, each XPC service has a unique handler function as the entry point, and some XPC system service binaries contain multiple XPC services. Thus, the number of XPC entry points identified by iService is slightly larger than the number of XPC service binaries. As for the NSXPC service, since its entry points are the class methods of its exported objects, the number of NSXPC entry points is multiple times the number of NSXPC services. As is shown in Table 1, iService identified 185 NSXPC services and 436 NSXCP entry points from the binaries, consistent with the above insight.

5.3.2 ***Sensitive Operation Identification.*** iService identified a total of 431 sensitive operations, of which 29 are used to modify file permissions and ownership, 127 are used to access system and user files, 37 are used to manage user and application preferences, and 194 are used to manage processes, as shown in Table 1. Since many sensitive operations used to manage processes appear in pairs, e.g., spawning a child process via NSTask requires two sensitive operations (i.e., `setLaunchPath` and `setArguments`) at the same time. Hence the number of process management operations is the highest among the four categories of sensitive operations.

In addition, some system services authenticate the requester's identity and dispatch the request to the objects implemented in private libraries. Therefore, some sensitive operations are in private libraries rather than service binaries. For example, the core logic of the motivating example is implemented in the private library, which contains sensitive operations in Lines 17 and 18. iService analyzes private libraries by using external functions in services as entry points, thereby finding 23 sensitive operations in private libraries. However, as discussed in Section 5.2.3, after macOS BigSur, private libraries have been combined into the dyld shared cache, which makes iService unable to find sensitive operations in private libraries in macOS 11.4 and 12.4, resulting in false negatives.

5.3.3 ***Controllable Operation Identification.*** As shown in Table 1, iService identifies a total of 87 controllable sensitive operations, of which the key parameters are dependent on the IPC input. The number of controllable sensitive operations is gradually reduced due to two main reasons. On the one hand, Apple repaired some vulnerabilities by removing unnecessary sensitive operations, thereby reducing the number of controllable sensitive operations. On the other hand, the data dependence analysis performed by iService produces false negatives. For example, the alias analysis adopted produces over-approximate alias set, which would cause a state explosion. iService then cannot handle all those states, which results in false negatives of controllable sensitive operations. In addition, failure to identify sensitive operations in the dyld shared cache leads to false negatives here as well.

5.4 Protection Evaluation.

5.4.1 ***Entitlement Verification.*** While analyzing permission checks, iService found three different ways to extract the entitlement, which are prone to misuse and result in the permission check bypassing. The three types of extractions are as follows:

(1) Get the entitlement directly from the incoming (NS)XPC connection object;

Table 3: Scores and serverity of vulnerabilities.

Vulnerable Service	score[1]	severity[2]	CVE ID
osanalyticshelper	1	7.8	CVE-2021-30774
fbahelperd	3	5.5	CVE-2019-8521
fbahelperd	2	7.0	CVE-2019-8565
timemachinehelper	1	7.8	CVE-2019-8513
timemachinehelper	3	5.5	CVE-2019-8530

1: Score is computed through the level weight method in Section 3.4.3.
2: Severity is provided by National Vulnerability Database (NVD).

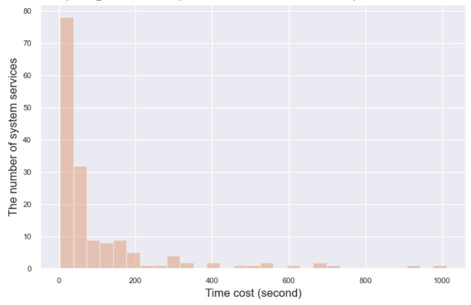


Figure 6: Analysis time cost distribution histogram.

(2) Obtain the target process through the audit token to extract the entitlement;
(3) Get the target process by PID and get its entitlement.

The first and second are property permission checks belonging to Level 4 (§ 3.4.2). The third one would be bypassed using race condition because it uses PID to locate the process. For example, the service `fud` in Table 2 obtains the entitlement of the target process via `copyEntitlementsForPid`, which allows malicious programs to bypass the authentication. Apple fixed it by changing the check to an audit token-based check.

5.4.2 ***Restrictions for String Inputs.*** iService found three restrictions on string parameters in sensitive input validations:

(1) C-based API (i.e., `strncmp`) to compare strings as expected;
(2) Objective-C API (i.e., isEqualToString) to determine if the string is as expected;
(3) `hasPrefix` method to determine whether the string prefix meets the requirements.

The first and second are Level 4 input validation. The third detection method is Level 3, which can be bypassed via path traversal. The problem exists in the 0-day bugs found (§ 5.6.1).

5.4.3 ***Evaluate Scoring Mechanism.*** To evaluate the effectiveness of iService's scoring mechanism, we compare the scores calculated for the five discovered CVEs to the severity provided by NVD, as shown in Table 3. The trend of the two is consistent, i.e., for the vulnerabilities more likely to exploit calculated by iService, the severity provided by NVD is higher. For example, CVE-2019-8521 has a higher score than other CVEs due to multiple Level 3 sensitive input validations on its utilization path, and the severity provided by NVD is also the lowest among several CVEs.

5.5 Performance

The distribution of time costs for analyzing system services with sensitive operations is shown in Figure 6. For nearly 90% of services, iService analyzed them within 4 minutes. The main time cost is in

```
-[FBAPrivilegedDaemon copyLogFiles:]
 1 if ([src hasPrefix:@"/Library/Logs"] || [src hasPrefix:@"/var/log"]){
 2   if (![self canModifyPath:dst]) {
 3     result[src] = [NSString stringWithFormat:@"Invalid
 4       destination: %@", dst];
 5   } else {
 6     result[src] = @"File must be copied from a log directory";
 7   }
 8 }
   ... // Skip some complex code
 9 [fileMg copyItemAtPath:src toPath:dst error:&err]
10 [self fixPermissionsOfURL:dst recursively:1]

-[FBAPrivilegedDaemon canModifyPath:]
11 if ([dst hasPrefix:@"/var/folders/"] ||
12    [dst hasPrefix:@"/private/var/"] || [dst hasPrefix:@"/tmp/"]) {
13   return TRUE;
14 } else {
15   return [dst rangeOfString:@"Library/Caches/
16     com.apple.appleseed.FeedbackAssistant"] != 0;
17 }
```

Figure 7: The critical code snippet in the confused deputy of CVE-2019-8521, where the binary is lifted to the Objective-C source code, e.g., the function call at Line 9 is the result of resolving `objc_msgSend(fileMg, "copyItemAtPath:ToPath:error:", src, dst, &err)`.

the function dataflow construction, which is a one-time effort. Neo4j generated overhead in merging and continuously updating the graph database, especially for services with complex functions and calling relationships. After such a one-time effort, graph database-based data dependence analysis and protection extraction can be completed in seconds.

5.6 Case Study

5.6.1 ***Case-1: CVE-2019-8521.*** The confused deputy in Figure 1 is located in the feedback assistant service, which can be abused to perform arbitrary file overwriting and further gain root privileges [6]. It performs the sensitive operation of moving files (Line 9) and the key parameters are string variables `src` and `dst`, which represent the source file and the destination path, respectively. The service restricts src and `dst` by checking their prefix via `hasPrefix` at Lines 1, 11, and 12. However, such prefix-path matching can be bypassed via path traversal, such as `/var/log/../../AnyPath`. Therefore, malicious programs can copy crafted files to the existing system file path for overwriting, thereby obtaining root privileges.

iService identifies such weak input validations via sensitive operation-oriented data dependence analysis. Specifically, it first resolves indirect calls to recover the calling relationship at the call site (Line 2) and identify the sensitive operation (Line 9). After that, iService extracts data dependencies between the IPC input and parameters used in file copying operation, i.e., `src` and `dst`. Then, it identifies statements in Lines 1, 2, 11, and 12 as sensitive input validations based on the data dependencies. Finally, iService finds that the service only relies on prefix matching to restrict `src` and funcdst, which can be bypassed, thus reporting a confused deputy.

5.6.2 ***Case-2: CVE-2019-8530.*** Figure 8 depicts the critical code snippet of the confused deputy in the Time Machine module, which can be abused for arbitrary file overwriting [3]. The vulnerability is caused by a lack of input validation for the key parameter of the task arguments setting operations in Line 16, which is used to set the command parameter of the NSTask-spawned child process. The child process executes `tmdiagnose`, generates hundreds of megabytes of system diagnostic information, and dumps it into

```
 1 txt = objc_msgSend(dir, "URLByAppendingPathComponent:",
 2          CFSTR("DETimeMachineExtension-error.txt"));
 3 tmd = CFSTR("/usr/bin/tmdiagnose");
 4 args[0] = CFSTR("-w");
 5 args[1] = CFSTR("-r");
 6 args[2] = CFSTR("-f");
 7 args[3] = objc_msgSend(dir, "path");
 8 argsArray = objc_msgSend(&OBJC_CLASS___NSArray,
 9                "arrayWithObjects:count:", args, 4LL);
10 fd = objc_msgSend(&OBJC_CLASS___NSFileHandle,
11        "fileHandleForWritingToURL:error:", txt, &err);
12 if (fd){
13   task = objc_msgSend(&OBJC_CLASS___NSTask, "alloc");
14   _task = objc_msgSend(task, "init");
15   objc_msgSend(_task, "setLaunchPath:", tmd);
16   objc_msgSend(_task, "setArguments:", argsArray);
     ... // Set other configures
17   objc_msgSend(_task, "launch");
```

Figure 8: The critical code snippet of CVE-2019-8530, where variables highlighted in yellow represent the dataflow from the input (dir) to sensitive operations (Line 16).

the directory dir specified by the input. Since dir is not validated, malicious programs can write diagnostic information to arbitrary directories, allowing DoS attacks, such as fulfilling a disk.

iService identifies data dependencies between input dir and argsArray in Line 16. Specifically, iService performs field-sensitive dataflow analysis, and constructs the dataflow between function parameters and the return value in Line 8. After that, iService found that dir could control the parameter of task arguments setting operation and there is no validations to restrict the value of dir. Therefore, iService reported a confused deputy.

6 DISCUSSION

6.1 Limitation in Reverse Engineering.

iService failed to analyze some system services where entry points could be identified, due to the limitation in reverse engineering. On the one hand, since iService uses IDA to lift the IR, the extent to which iService reverse the binary depends on the IR processing power of IDA. When IDA cannot properly disassemble and generate IR, iService cannot work as well. And the errors generated by the IDA would also affect the type recovery and call relationship identification of iService. On the other hand, the update of the OS would introduce new compiler features into binaries, which cause iService to encounter new cases that require additional processing on the latest version. In addition, the dyld shared cache is also updated and its format is changed. Resulting in more reverse engineering being required to parse it. Therefore, more reverse engineering on AppleOS is left as future work.

6.2 Comparison to Kobold

Kobold is the only existing work looking into confused deputies in AppleOS. It leverages black box fuzzing and manual analysis to detect confused deputies caused by a lack of access control. Comparing the two on the methodology level, iService outperforms Kobold due to its problem scope and code coverage. iService can handle input validations and numerous kinds of permission checks, while Kobold only focuses on one type of permission check called entitlement check. Furthermore, iService provides the method based on static analysis with high code coverage, while Kobold is based on black-box fuzzing with limited code coverage.

As for bug detection, iService finds five CVEs, and it can be inferred that Kobold can find none of them due to the two limitations discussed above. CVE-2019-8521 and CVE-2019-8565 correspond to permission checks via PID, which Kobold does not cover due to the problem scope Issue. Triggering CVE-2021-30774, CVE-2019-8513, and CVE-2019-8530 requires to understand logic flaws, e.g., complex string input with specific meaning, which can hardly be achieved by black-box fuzzing used in Kobold.

7 RELATED WORK

Previous work in AppleOS. Recent years, works on AppleOS mainly focus on bug detection of the AppleOS driver and IPC mechanism. iDEA [10] detected conditional competition and out-of-bound read and write in the kernel driver using static analysis. SyzGen [14] automated the generation of syscall specifications for closed-source macOS drivers and performed interface-aware fuzzing based on it. As for the IPC mechanism, Min Zheng et al. [34] summarized the (Mach) Port-oriented Programming (POP) attack to exploit kernel memory corruption vulnerability, and proposed the Port Ultra-SHield (PUSH) to defend against POP attack. In userspace, IPC mechanism may be a breakthrough in Apple's access control [9, 11, 12, 17]. Kobold [16] performed a black-box fuzzing with manual analysis to detect confused deputies lacking entitlement checks in iOS. However, there is still no systematic research for confused deputy detection in AppleOS.
Confused deputy detection in Android. Existing work [1, 9, 19, 20, 22, 23, 31] discovers and utilizes misconfigured permission checks of Android system services based on statics source code analysis.. Buzzer [13] and FANS [26] leverage fuzzing to detect memory corruption caused by a lack of input validations, while Invetter [33] uses a learning-based recognition method relying on variable names to locate problematic sensitive input validations.

8 CONCLUSION

In this paper, we propose iService, a static analyzer to detect confused deputies and evaluate the impact of them. iService adopt a top-down type propagation to resolve function calls, and perform sensitive operation-oriented data dependence analysis to extract permission checks and input validations. It evaluates the protections to report confused deputies . iService was applied to 439 system services from four AppleOS versions and discovered 11 confused deputies. Five of them are 0-day bugs, which are assigned with CVE numbers, and others are 1-day bugs. We reported them to Apple and all of them have been fixed.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable feedback to improve our manuscript. We especially thank Zhi Zhou (CodeColorist) for his contribution to our work. In addition, this work is partially supported by Shanghai Pujiang Program (No. 22PJ1405700) and the National Key Research and Development Program of China (No. 2020AAA0107803, No. 2021YFB3101402, and No. 2021QY2333).

REFERENCES

[1] Yousra Aafer, Guanhong Tao, Jianjun Huang, Xiangyu Zhang, and Ninghui Li. 2018. Precise Android API protection mapping derivation and reasoning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1151–1164.

[2] Apple Inc. 2016. Elevating Privileges Safely - Apple Developers. https://developer.apple.com/library/archive/documentation/Security/Conceptual/SecureCodingGuide/Articles/AccessControl.html. [Online; accessed 8-June-2022].

[3] Apple Inc. 2019. About the security content of macOS Mojave 10.14.4. https://support.apple.com/en-us/HT209600. [Online; accessed 28-June-2022].

[4] Apple Inc. 2021. About the security content of iOS 12.2. https://support.apple.com/en-us/HT209599. [Online; accessed 28-June-2022].

[5] Apple Inc. 2021. About the security content of iOS 14.7 and iPadOS 14.7. https://support.apple.com/en-us/HT212601. [Online; accessed 28-June-2022].

[6] Apple Inc. 2021. About the security content of macOS Big Sur 11.5. https://support.apple.com/en-us/HT212602. [Online; accessed 28-June-2022].

[7] Apple Inc. 2021. About the security content of tvOS 14.7. https://support.apple.com/en-us/HT212604. [Online; accessed 28-June-2022].

[8] Aronskaka. 2020. Job(s) Bless Us! Privileged Operations on macOS. https://speakerdeck.com/vashchenko/job-s-bless-us-privileged-operations-on-macos?slide=2. [Online; accessed 8-June-2022].

[9] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. 2012. Pscout: analyzing the android permission specification. In *Proceedings of the 2012 ACM conference on Computer and communications security*. 217–228.

[10] Xiaolong Bai, Luyi Xing, Min Zheng, and Fuping Qu. 2020. idea: Static analysis on the security of apple kernel drivers. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 1185–1202.

[11] Ian Beer. 2015. Auditing and Exploiting Apple IPC. Accessed: 2022-06-01.

[12] Tyler Bohan. 2019. OSX XPC Revisited - 3rd Party Application Flaws. Accessed: 2022-06-01.

[13] Chen Cao, Neng Gao, Peng Liu, and Ji Xiang. 2015. Towards analyzing the input validation vulnerabilities associated with android system services. In *Proceedings of the 31st Annual Computer Security Applications Conference*. 361–370.

[14] Weiteng Chen, Yu Wang, Zheng Zhang, and Zhiyun Qian. 2021. SyzGen: Automated Generation of Syscall Specification of Closed-Source macOS Drivers. In *ACM CCS*.

[15] Zhui Deng, Brendan Saltaformaggio, Xiangyu Zhang, and Dongyan Xu. 2015. iris: Vetting private api abuse in ios applications. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 44–56.

[16] Luke Deshotels, Costin Carabas, Jordan Beichler, Răzvan Deaconescu, and William Enck. 2020. Kobold: Evaluating decentralized access control for remote NSXPC methods on iOS. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1056–1070.

[17] Luke Deshotels, Razvan Deaconescu, Costin Carabas, Iulia Manda, William Enck, Mihai Chiroiu, Ninghui Li, and Ahmad-Reza Sadeghi. 2018. iOracle: Automated evaluation of access control policies in iOS. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. 117–131.

[18] Manuel Egele, Christopher Kruegel, Engin Kirda, and Giovanni Vigna. 2011. PiOS: Detecting Privacy Leaks in iOS Applications.. In *NDSS*. 177–183.

[19] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. 2011. Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security*. 627–638.

[20] Adrienne Porter Felt, Helen J Wang, Alexander Moshchuk, Steve Hanna, and Erika Chin. 2011. Permission Re-Delegation: Attacks and Defenses.. In *USENIX security symposium*, Vol. 30. 88.

[21] Sigmund Albert Gorski, Benjamin Andow, Adwait Nadkarni, Sunil Manandhar, William Enck, Eric Bodden, and Alexandre Bartel. 2019. Acminer: Extraction and analysis of authorization checks in android's middleware. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*. 25–36.

[22] Sigmund Albert Gorski III and William Enck. 2019. Arf: identifying re-delegation vulnerabilities in android system services. In *Proceedings of the 12th conference on security and privacy in wireless and mobile networks*. 151–161.

[23] Sigmund Albert Gorski III, Seaver Thorn, William Enck, and Haining Chen. 2022. FRED: Identifying File Re-Delegation in Android System Services. (2022).

[24] Samuel Groß. 2018. Don't Trust the PID! Stories of a simple logic bug and where to find it. https://saelo.github.io/presentations/warcon18_dont_trust_the_pid.pdf. [Online; accessed 8-June-2022].

[25] Hex-Rays. 2022. IDA Pro: A powerful disassembler and a versatile debugger. https://www.hex-rays.com/products/ida. [Online; accessed 28-June-2022].

[26] Baozheng Liu, Chao Zhang, Guang Gong, Yishun Zeng, Haifeng Ruan, and Jianwei Zhuge. 2020. {FANS}: Fuzzing Android Native System Services via Automated Interface Analysis. In *29th USENIX Security Symposium (USENIX Security 20)*. 307–323.

[27] Neo4j Team. 2022. NEO4J GRAPH DATA PLATFORM: Blazing-Fast Graph, Petabyte Scale. https://neo4j.com. [Online; accessed 28-June-2022].

[28] Thomas Reps, Susan Horwitz, and Mooly Sagiv. 1995. Precise interprocedural dataflow analysis via graph reachability. In *Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 49–61.

[29] Thomas Reps, Mooly Sagiv, and Susan Horwitz. 1994. *Interprocedural dataflow analysis via graph reachability*. Datalogisk Institut, Københavns Universitet.

[30] Wojciech. 2020. Abusing and Securing XPC in macOS apps. https://objectivebythesea.org. [Online; accessed 8-June-2022].

[31] Xiaobo Xiang, Ren Zhang, Hanxiang Wen, Xiaorui Gong, and Baoxu Liu. 2021. Ghost in the Binder: Binder Transaction Redirection Attacks in Android System Services. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 1581–1597.

[32] Fabian Yamaguchi, Nico Golde, Daniel Arp, and Konrad Rieck. 2014. Modeling and discovering vulnerabilities with code property graphs. In *2014 IEEE Symposium on Security and Privacy*. IEEE, 590–604.

[33] Lei Zhang, Zhemin Yang, Yuyu He, Zhenyu Zhang, Zhiyun Qian, Geng Hong, Yuan Zhang, and Min Yang. 2018. Invetter: Locating insecure input validations in android services. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1165–1178.

[34] Min Zheng, Xiaolong Bai, Yajin Zhou, Chao Zhang, and Fuping Qu. 2021. POP and PUSH: Demystifying and Defending against (Mach) Port-oriented Programming.. In *NDSS*.

A SENSITIVE OPERATION COLLECTION

iService collects a total of 40 C-APIs and 94 Objective-C APIs as sensitive operations according to Apple's public documents, as shown in Table 4.

Table 4: Sensitive operation collection.

Category	API Type	Function
Manipulate file permissions and ownership	C-based	_fchown
		_fchmod
	Objective-C-based	-[NSFileManager changeFileAttributes:atPath:]
		-[NSFileManager setAttributes:ofItemAtPath:error:]
Manipulate system and user files	C-based	_dlopen
	Objective-C-based	-[NSFileManager copyPath:toPath:handler:]
		-[NSFileManager movePath:toPath:handler:]
		-[NSFileManager removeFileAtPath:handler:]
		-[NSFileManager removeItemAtURL:error:]
		-[NSFileManager removeItemAtPath:error:]
		-[NSFileManager linkPath:toPath:handler:]
		-[NSFileManager copyItemAtURL:toURL:error:]
		-[NSFileManager copyItemAtPath:toPath:error:]
		-[NSFileManager moveItemAtURL:toURL:error:]
		-[NSFileManager moveItemAtPath:toPath:error:]
		-[NSFileManager createSymbolicLinkAtURL:...:error:]
		-[NSFileManager createSymbolicLinkAtPath:...:error:]
		-[NSFileManager createDirectoryAtURL:...:attributes:error:]
		-[NSFileManager linkItemAtURL:toURL:error:]
		-[NSFileManager linkItemAtPath:toPath:error:]
		-[NSFileManager createFileAtPath:contents:attributes:]
		-[NSString writeToFile:atomically:encoding:error:]
		-[NSString writeToURL:atomically:encoding:error:]
		+[NSString stringWithContentsOfFile:encoding:error:]
		-[NSString initWithContentsOfFile:encoding:error:]
		+[NSString stringWithContentsOfFile:usedEncoding:error:]
		-[NSString initWithContentsOfFile:usedEncoding:error:]
		+[NSString stringWithContentsOfURL:encoding:error:]
		-[NSString initWithContentsOfURL:encoding:error:]
		+[NSString stringWithContentsOfURL:usedEncoding:error:]
		-[NSString initWithContentsOfURL:usedEncoding:error:]
		+[NSDictionary dictionaryWithContentsOfURL:error:]
		+[NSDictionary dictionaryWithContentsOfURL:]
		-[NSDictionary initWithContentsOfURL:error:]
		-[NSDictionary initWithContentsOfURL:]
		+[NSDictionary dictionaryWithContentsOfFile:]
		-[NSDictionary initWithContentsOfFile:]
		-[NSDictionary writeToURL:error:]
		-[NSDictionary writeToURL:atomically:]
		-[NSDictionary writeToFile:atomically:]
		+[NSArray arrayWithContentsOfFile:]
		+[NSArray arrayWithContentsOfURL:]
		+[NSArray arrayWithContentsOfURL:error:]
		-[NSArray initWithContentsOfURL:error:]
		-[NSArray writeToFile:atomically:]
		-[NSArray writeToURL:atomically:]
		-[NSArray writeToURL:error:]
		+[NSData dataWithContentsOfFile:]
		+[NSData dataWithContentsOfFile:options:error:]
		+[NSData dataWithContentsOfURL:]
		+[NSData dataWithContentsOfURL:options:error:]
		-[NSData initWithContentsOfFile:]
		-[NSData initWithContentsOfFile:options:error:]

		-[NSData initWithContentsOfURL:]
		-[NSData initWithContentsOfURL:options:error:]
		-[NSData initWithContentsOfMappedFile:]
		+[NSData dataWithContentsOfMappedFile:]
		-[NSData writeToFile:atomically:]
		-[NSData writeToFile:options:error:]
		-[NSData writeToURL:atomically:]
		-[NSData writeToURL:options:error:]
Manage user and application preference	C-based	_CFPreferencesCopyAppValue
		_CFPreferencesCopyValue
		_CFPreferencesSetValue
		_CFPreferencesSetMultiple
		_CFPreferencesSetAppValue
Manipulate system and user files	C-based	_system
		_execl
		_execlp
		_execle
		_execv
		_execvp
		_execvP
		_sandbox_init
		_sandbox_init_with_parameters
		_posix_spawn
		_posix_spawnp
		_LSOpenApplication
		_LSOpenURLsWithRole
		_LSOpenFSRef
		_LSOpenCFURLRef
		_LSOpenFromURLSpec
		_CFBundleCreate
		_CFBundleCreateBundlesFromDirectory
		_CFBundleLoadExecutable
		_CFBundleLoadExecutableAndReturnError
		_SMJobSubmit
		_SMJobBless
	Objective-C-based	+[NSTask launchedTaskWithLaunchPath:arguments:]
		-[NSTask setLaunchPath:]
		-[NSTask setArguments:]
		-[NSTask launch]
		-[NSTask launchAndReturnError:]
		+[NSTask launchedTaskWithExecutableURL:...:]
		-[NSUserScriptTask initWithURL:error:]
		-[NSUserScriptTask setScriptURL]
		-[NSUserScriptTask executeWithCompletionHandler:]
		-[NSUserAppleScriptTask executeWithAppleEvent:...:]
		-[NSUserUnixTask executeWithArguments:completionHandler:]
		+[NSBundle bundleWithURL:]
		+[NSBundle bundleWithPath:]
		-[NSBundle initWithURL:]
		-[NSBundle initWithPath:]
		-[NSBundle load]
		-[NSBundle loadAndReturnError:]
		+[NSInvocation invocationWithMethodSignature:]
		-[NSInvocation invoke]
		-[NSInvocation invokeWithTarget:]
		-[NSInvocation setArgument:atIndex:]

		+[NSExpression expressionWithFormat:]
		+[NSExpression expressionWithFormat:argumentArray:]
		+[NSExpression expressionWithFormat:arguments:]
		-[NSExpression expressionValueWithObject:context:]
		-[NSWorkspace openURL:options:configuration:error:]
		-[NSWorkspace openURLs:withApplicationAtURL:...:error:]
		-[NSWorkspace openFile:]
		-[NSWorkspace openFile:withApplication:]
		-[NSWorkspace openFile:withApplication:andDeactivate:]
		-[NSWorkspace openFile:fromImage:at:inView:]