Atlassian: osquery

Security Assessment

January 24, 2022

Prepared for:
Gui Vieiro
Florian Ruechel
William Shoemaker
Atlassian

Prepared by:
Anders Helsing
Emilio López

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 80+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

This report is considered by Trail of Bits to be public information; it is licensed to Atlassian under the terms of the project statement of work and has been made public at Atlassian's request. Material within this report may not be reproduced or distributed in part or in whole without the express written permission of Trail of Bits.

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and mutually agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As such, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Table of Contents

Executive Summary

Engagement Overview

Atlassian engaged Trail of Bits to review the security of the open-source osquery project. From January 3 to January 24, 2022, a team of two consultants conducted a security review of the client-provided source code, with six person-weeks of effort. Details of the project's timeline, test targets, and coverage are provided in subsequent sections of this report.

Project Scope

Our testing efforts were focused on the identification of flaws that could result in a compromise of confidentiality, integrity, or availability of the target system. We conducted this audit with full knowledge of the target system, including access to the source code and documentation. We manually reviewed the project to gain a deeper understanding of the codebase, and we used static analysis tools such as `clang-tidy`, `cppcheck`, and `scan-build`.

Summary of Findings

The audit uncovered significant flaws that could impact system confidentiality, integrity, or availability. A summary of the findings and details on notable findings are provided below.

EXPOSURE ANALYSIS

Severity	*Count*
High	2
Medium	8
Low	4
Informational	3
Undetermined	0

CATEGORY BREAKDOWN

Category	*Count*
Denial of Service	9
Data Validation	3
Timing	2
Configuration	1
Patching	1
Undefined Behavior	1

Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

- **TOB-ATL-1, TOB-ATL-2**
  The project uses a large number of dependencies but lacks an established process for tracking vulnerabilities within them. Without such a process, the osquery agent could lack the necessary patches for published vulnerabilities. This problem is further compounded by the fact that the dependency code is executed within the same security context as the agent (i.e., “root”) and that it is used extensively to parse untrusted data.

- **TOB-ATL-5, TOB-ATL-8, TOB-ATL-9, TOB-ATL-12, TOB-ATL-13, TOB-ATL-14, TOB-ATL-17**
  A large number of issues pertain to code that enables attackers to create conditions that would either stall or crash the osquery agent. The project contains numerous corner cases that can be used to create conditions that stall the agent, which indicates that the architecture is insufficiently resilient in this respect. As of this writing, a watchdog is used to monitor the agent. When triggered, the watchdog terminates and restarts the agent, marks the offending query, and waits 24 hours before rerunning the query to ensure that other queries can continue to execute. However, the watchdog triggers on excessive CPU or memory use and does not catch hangs caused by blocking syscalls.

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan@trailofbits.com

Mary O'Brien, Project Manager
mary.obrien@trailofbits.com

The following engineers were associated with this project:

Anders Helsing, Consultant
anders.helsing@trailofbits.com

Emilio López, Consultant
emilio.lopez@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
December 8, 2021	Pre-project kickoff call
January 18, 2022	Status update meeting #1
January 24, 2022	Delivery of report draft
February 14, 2022	Delivery of final report

Project Goals

The engagement was scoped to provide a security assessment of the open-source osquery project. Specifically, we sought to answer the following non-exhaustive list of questions:

- Are there any issues related to insufficient data validation in the code that ingests attacker-controlled data? Could these issues be used to stop osquery from working or to escalate an attacker's privileges to those of the osquery agent?
- How does osquery recover from a crash or stalled state?
- Are the files used by osquery sufficiently protected from modification by an attacker?
- Is the inter-process communication channel sufficiently protected from less-privileged users?
- Is there any separation between code running within the high-security context and code parsing attacker-controlled data?
- How are vulnerabilities in dependencies tracked?

Project Targets

The engagement involved a review and testing of the following target.

Open Source

Repository	https://github.com/osquery/osquery
Version	`4274d3bfaedbcf7fa2b7b225abc2c2fb1b1b2e5a`
Type	C/C++
Platforms	Linux, macOS, and Windows

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches and their results include the following:

- A static analysis and manual review of the osquery core code
- A manual review of the osquery table code—including `atom_packages`, `browser_firefox`, `browser_opera`, `browser_plugins`, `elf_info`, `carbon_black`, `carves`, `chrome_extensions`, `docker`, `extended_attributes`, `lxd`, `office_mru`, `sip_config`, `time`, `xprotect`, and `yara`—with a special emphasis on table implementations handling user data
- A review of the installed state for the Linux, macOS, and Windows operating systems
- A review of recently disclosed CVEs in dependency code libraries used by the osquery codebase

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. During this project, we were unable to perform comprehensive testing of the following system elements, which may warrant further review:

- Code that is specific to the Windows implementation
- The Thrift inter-process communication channel
    - This channel cannot be used unless an attacker already has the same privileges as the osquery agent.
- Tables that do not read data
    - The focus of the audit is how an attacker can influence the osquery agent by manipulating data read by tables; therefore, we covered only the tables that read such data.

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	We identified an issue concerning integer truncation (TOB-ATL-15), one code-quality issue, and one issue concerning a missing overflow check (TOB-ATL-17).	**Moderate**
Auditing	The system uses a structured log mechanism to monitor events.	**Satisfactory**
Authentication / Access Controls	There is no security boundary between the library code and the agent; therefore, the code running with high privileges has a large attack surface. However, the permissions of the inter-process communication channel are sufficient to protect the system from misuse.	**Moderate**
Complexity Management	The code is well structured, organized into discrete components, and easy to follow.	**Satisfactory**
Configuration	The permissions assigned to installed files and directories prevent attackers from tampering with the system configuration. Using additional compiler mitigations would increase the difficulty of exploiting memory corruption issues.	**Satisfactory**
Cryptography and Key Management	The system provides options to ensure the enrollment key is not left in logs on deployment.	**Satisfactory**

Data Handling	While there are attempts to mitigate the impact of parsing bad data, these attempts are often insufficient or unused.	Weak
Documentation	The documentation is extensive and up to date.	Strong
Maintenance	There is no process for regularly updating dependencies to mitigate the risk of vulnerabilities within them.	Missing
Memory Safety and Error Handling	While much of the core code uses available mechanisms in C++ to provide memory safety, the dependency code may have issues. While there is a watchdog, the lack of a heartbeat function limits its effectiveness.	Moderate
Testing and Verification	A set of unit tests is available to verify the functionality of components, but several tables remain untested. The integration test suite can also be improved, as many of the tests currently perform only basic sanity checks on their corresponding tables.	Moderate

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Project dependencies are not monitored for vulnerabilities	Patching	**High**
2	No separation of privileges when executing dependency code	Configuration	**High**
3	No limit on the amount of information that can be read from the Firefox add-ons table	Denial of Service	**Low**
4	The SIP status on macOS may be misreported	Data Validation	**Informational**
5	The OpenReadableFile function can hang on reading a file	Denial of Service	**Medium**
6	Methods in POSIX PlatformFile class are susceptible to race conditions	Timing	**Low**
7	No limit on the amount of data that parsePlist can parse	Denial of Service	**Low**
8	The parsePlist function can hang on reading certain files	Denial of Service	**Medium**
9	The parseJSON function can hang on reading certain files on Linux and macOS	Denial of Service	**Medium**
10	No limit on the amount of data read or expanded from the Safari extensions table	Denial of Service	**Low**
11	Extended attributes table may read uninitialized or out-of-bounds memory	Timing	**Medium**

12	The readFile function can hang on reading a file	Denial of Service	**Medium**
13	The POSIX PlatformFile constructor may block the osquery thread	Denial of Service	**Medium**
14	No limit on the amount of data the Carver::blockwiseCopy method can write	Denial of Service	**Medium**
15	The carves table truncates large file sizes to 32 bits	Data Validation	**Informational**
16	The time table may not null-terminate strings correctly	Undefined Behavior	**Informational**
17	The elf_info table can crash the osquery agent	Data Validation	**Medium**

Detailed Findings

1. Project dependencies are not monitored for vulnerabilities	
Severity: **High**	Difficulty: **Medium**
Type: Patching	Finding ID: TOB-ATL-1
Target: osquery table dependencies	

Description

The osquery project depends on a large number of dependencies to realize the functionality of the existing tables. They are included as Git submodules in the project. The build mechanism of each dependency has been rewritten to suit the specific needs of osquery (e.g., so that it has as few dynamically loaded libraries as possible), but there is no process in place to detect published vulnerabilities in the dependencies. As a result, osquery could continue to use code with known vulnerabilities in the dependency projects.

Exploit Scenario

An attacker, who has gained a foothold on a machine running osquery, leverages an existing vulnerability in a dependency to exploit osquery. He escalates his privileges to those of the osquery agent or carries out a denial-of-service attack to block the osquery agent from sending data.

Recommendations

Short term, regularly update the dependencies to their latest versions.

Long term, establish a process within the osquery project to detect published vulnerabilities in its dependencies.

2. No separation of privileges when executing dependency code	
Severity: **High**	Difficulty: **Medium**
Type: Configuration	Finding ID: TOB-ATL-2
Target: osquery table dependencies	

Description

In several places in the codebase, the osquery agent realizes the functionality of a table by invoking code in a dependency project. For example, the `yara` table is implemented by invoking code in `libyara`. However, there is no separation of privileges or sandboxing in place when the code in the dependency library is called, so the library code executes with the same privileges as the osquery agent. Considering the project's numerous dependencies, this issue increases the osquery agent's attack surface and would exacerbate the effects of any vulnerabilities in the dependencies.

Exploit Scenario

An attacker finds a vulnerability in a dependency library that allows her to gain code execution, and she elevates her privileges to that of the osquery agent.

Recommendations

Short term, regularly update the dependencies to their latest versions.

Long term, create a security barrier against the dependency library code to minimize the impact of vulnerabilities.

3. No limit on the amount of information that can be read from the Firefox add-ons table

Severity: **Low**	Difficulty: **Low**
Type: Denial of Service	Finding ID: TOB-ATL-3
Target: `osquery/tables/applications/browser_firefox.cpp`	

Description

The implementation of the Firefox add-ons table has no limit on the amount of information that it can read from JSON files while enumerating the add-ons installed on a user profile. This is because to directly read and parse the Firefox profile JSON file, the `parseJSON` implementation in the osquery agent uses `boost::property_tree`, which does not have this limit.

```
pt::ptree tree;
if (!osquery::parseJSON(path + kFirefoxExtensionsFile, tree).ok()) {
  TLOG << "Could not parse JSON from: " << path + kFirefoxExtensionsFile;
  return;
}
```

Figure 3.1: The `osquery::parseJSON` function has no limit on the amount of data it can read.

Exploit Scenario

An attacker crafts a large, valid JSON file and stores it on the Firefox profile path as `extensions.json` (e.g., in `~/Library/Application Support/Firefox/Profiles/foo/extensions.json` on a macOS system). When osquery executes a query using the `firefox_addons` table, the `parseJSON` function reads and parses the complete file, causing high resource consumption.

Recommendations

Short term, enforce a maximum file size within the Firefox table, similar to the limits on other tables in osquery.

Long term, consider removing `osquery::parseJSON` and implementing a single, standard way to parse JSON files across osquery. The osquery project currently uses both `boost::property_tree` and RapidJSON libraries to parse JSON files, resulting in the use of different code paths to handle untrusted content.

4. The SIP status on macOS may be misreported

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-ATL-4
Target: `osquery/tables/system/darwin/sip_config.cpp`	

Description

If System Integrity Protection (SIP) is disabled on a Mac running osquery, the SIP configuration table might not report the correct value in the `enabled` column for the `config_flag: sip` row. For this misreporting to happen, extra flags need to be present in the value returned by `csr_get_active_config` and absent in the osquery `kRootlessConfigFlags` list. This is the case for the flags `CSR_ALLOW_ANY_RECOVERY_OS`, `CSR_ALLOW_UNAPPROVED_KEXTS`, `CSR_ALLOW_EXECUTABLE_POLICY_OVERRIDE`, and `CSR_ALLOW_UNAUTHENTICATED_ROOT` in `xnu-7195.141.2/bsd/sys/csr.h` (compare figure 4.1 and figure 4.3).

```
/* CSR configuration flags */
#define CSR_ALLOW_UNTRUSTED_KEXTS               (1 << 0)
#define CSR_ALLOW_UNRESTRICTED_FS               (1 << 1)
#define CSR_ALLOW_TASK_FOR_PID                  (1 << 2)
#define CSR_ALLOW_KERNEL_DEBUGGER               (1 << 3)
#define CSR_ALLOW_APPLE_INTERNAL                (1 << 4)
#define CSR_ALLOW_DESTRUCTIVE_DTRACE            (1 << 5) /* name deprecated */
#define CSR_ALLOW_UNRESTRICTED_DTRACE           (1 << 5)
#define CSR_ALLOW_UNRESTRICTED_NVRAM            (1 << 6)
#define CSR_ALLOW_DEVICE_CONFIGURATION          (1 << 7)
#define CSR_ALLOW_ANY_RECOVERY_OS               (1 << 8)
#define CSR_ALLOW_UNAPPROVED_KEXTS              (1 << 9)
#define CSR_ALLOW_EXECUTABLE_POLICY_OVERRIDE    (1 << 10)
#define CSR_ALLOW_UNAUTHENTICATED_ROOT          (1 << 11)
```

Figure 4.1: The CSR flags in `xnu-7159.141.2`

```
QueryData results;
csr_config_t config = 0;
csr_get_active_config(&config);

csr_config_t valid_allowed_flags = 0;
for (const auto& kv : kRootlessConfigFlags) {
  valid_allowed_flags |= kv.second;
}

Row r;
r["config_flag"] = "sip";
if (config == 0) {
  // SIP is enabled (default)
  r["enabled"] = INTEGER(1);
  r["enabled_nvram"] = INTEGER(1);
} else if ((config | valid_allowed_flags) == valid_allowed_flags) {
  // mark SIP as NOT enabled (i.e. disabled) if
  // any of the valid_allowed_flags is set
  r["enabled"] = INTEGER(0);
  r["enabled_nvram"] = INTEGER(0);
}
results.push_back(r);
```

Figure 4.2: How the SIP state is computed in `genSIPConfig`

```
// rootless configuration flags
// https://opensource.apple.com/source/xnu/xnu-3248.20.55/bsd/sys/csr.h
const std::map<std::string, uint32_t> kRootlessConfigFlags = {
    // CSR_ALLOW_UNTRUSTED_KEXTS
    {"allow_untrusted_kexts", (1 << 0)},
    // CSR_ALLOW_UNRESTRICTED_FS
    {"allow_unrestricted_fs", (1 << 1)},
    // CSR_ALLOW_TASK_FOR_PID
    {"allow_task_for_pid", (1 << 2)},
    // CSR_ALLOW_KERNEL_DEBUGGER
    {"allow_kernel_debugger", (1 << 3)},
    // CSR_ALLOW_APPLE_INTERNAL
    {"allow_apple_internal", (1 << 4)},
    // CSR_ALLOW_UNRESTRICTED_DTRACE
    {"allow_unrestricted_dtrace", (1 << 5)},
    // CSR_ALLOW_UNRESTRICTED_NVRAM
    {"allow_unrestricted_nvram", (1 << 6)},
    // CSR_ALLOW_DEVICE_CONFIGURATION
    {"allow_device_configuration", (1 << 7)},
};
```

Figure 4.3: The flags currently supported by osquery

Exploit Scenario

An attacker, who has gained a foothold with root privileges, disables SIP on a device running macOS and sets the `csr_config` flags to `0x3e7`. When building the response for the `sip_config` table, osquery misreports the state of SIP.

Recommendations

Short term, consider reporting SIP as disabled if any flag is present or if any of the known flags are present (e.g., if `(config & valid_allowed_flags) != 0`).

Long term, add support for reporting the raw flag values to the table specification and code so that the upstream server can make the final determination on the state of SIP, irrespective of the flags supported by the osquery daemon. Additionally, monitor for changes and add support for new flags as they are added on the macOS kernel.

5. The OpenReadableFile function can hang on reading a file

Severity: **Medium**	Difficulty: **Low**
Type: Denial of Service	Finding ID: TOB-ATL-5
Target: `osquery/filesystem/filesystem.cpp`	

Description

The `OpenReadableFile` function creates an instance of the `PlatformFile` class, which is used for reading and writing files. The constructor of `PlatformFile` uses the open syscall to obtain a handle to the file. The `OpenReadableFile` function by default opens the file using the `O_NONBLOCK` flag, but if `PlatformFile`'s `isSpecialFile` method returns true, it opens the file without using `O_NONBLOCK`. On the POSIX platform, the `isSpecialFile` method returns true for files in which `fstat` returns a size of zero. If the file to be read is a FIFO, the open syscall in the second invocation of `PlatformFile`'s constructor blocks the osquery thread until another thread opens the FIFO file to write to it.

```
struct OpenReadableFile : private boost::noncopyable {
 public:
  explicit OpenReadableFile(const fs::path& path, bool blocking = false)
      : blocking_io(blocking) {
    int mode = PF_OPEN_EXISTING | PF_READ;
    if (!blocking) {
      mode |= PF_NONBLOCK;
    }

    // Open the file descriptor and allow caller to perform error checking.
    fd = std::make_unique<PlatformFile>(path, mode);

    if (!blocking && fd->isSpecialFile()) {
      // A special file cannot be read in non-blocking mode, reopen in blocking
      // mode
      mode &= ~PF_NONBLOCK;
      blocking_io = true;
      fd = std::make_unique<PlatformFile>(path, mode);
    }
  }

 public:
  std::unique_ptr<PlatformFile> fd{nullptr};
  bool blocking_io;
};
```

Figure 5.1: The `OpenReadableFile` function can block the osquery thread.

Exploit Scenario

An attacker creates a special file, such as a FIFO, in a path known to be read by the osquery agent. When the osquery agent attempts to open and read the file, it blocks the osquery thread indefinitely, in effect making osquery unable to report the status to the server.

Recommendations

Short term, ensure that the file operations in `filesystem.cpp` do not block the osquery thread.

Long term, introduce a timeout on file operations so that a block does not stall the osquery thread.

References

- The Single Unix Specification, Version 2

6. Methods in POSIX PlatformFile class are susceptible to race conditions

Severity: **Low**	Difficulty: **Medium**
Type: Timing	Finding ID: TOB-ATL-6
Target: `osquery/filesystem/posix/fileops.cpp`	

Description

The POSIX implementation of the methods in the `PlatformFile` class includes several methods that return the current properties of a file. However, the properties can change during the lifetime of the file descriptor, so the return values of these methods may not reflect the actual properties. For example, the `isSpecialFile` method, which is used to determine the strategy for reading the file, calls the `size` method. However, the file size can change between the time of the call and the reading operation, in which case the wrong strategy for reading the file could be used.

```
bool PlatformFile::isSpecialFile() const {
  return (size() == 0);
}

static uid_t getFileOwner(PlatformHandle handle) {
  struct stat file;
  if (::fstat(handle, &file) < 0) {
    return -1;
  }
  return file.st_uid;
}

Status PlatformFile::isOwnerRoot() const {
  if (!isValid()) {
    return Status(-1, "Invalid handle_");
  }

  uid_t owner_id = getFileOwner(handle_);
  if (owner_id == (uid_t)-1) {
    return Status(-1, "fstat error");
  }

  if (owner_id == 0) {
    return Status::success();
  }
  return Status(1, "Owner is not root");
}

Status PlatformFile::isOwnerCurrentUser() const {
  if (!isValid()) {
    return Status(-1, "Invalid handle_");
  }

  uid_t owner_id = getFileOwner(handle_);
  if (owner_id == (uid_t)-1) {
    return Status(-1, "fstat error");
  }

  if (owner_id == ::getuid()) {
    return Status::success();
  }

  return Status(1, "Owner is not current user");
}

Status PlatformFile::isExecutable() const {
  struct stat file_stat;
  if (::fstat(handle_, &file_stat) < 0) {
    return Status(-1, "fstat error");
  }

  if ((file_stat.st_mode & S_IXUSR) == S_IXUSR) {
    return Status::success();
  }

  return Status(1, "Not executable");
}

Status PlatformFile::hasSafePermissions() const {
  struct stat file;
  if (::fstat(handle_, &file) < 0) {
    return Status(-1, "fstat error");
  }

  // We allow user write for now, since our main threat is external
  // modification by other users
  if ((file.st_mode & S_IWOTH) == 0) {
    return Status::success();
  }

  return Status(1, "Writable");
}
```

Figure 6.1: The methods in `PlatformFile` could cause race issues.

Exploit Scenario

A new function is added to osquery that uses `hasSafePermissions` to determine whether to allow a potentially unsafe operation. An attacker creates a file that passes the `hasSafePermissions` check, then changes the permissions and alters the file contents before the file is further processed by the osquery agent.

Recommendations

Short term, refactor the operations of the *relevant* `PlatformFile` class methods to minimize the race window. For example, the only place `hasSafePermissions` is currently used is in the `safePermissions` function, in which it is preceded by a check that the file is owned by root or the current user, which eliminates the possibility of an adversary using the race condition; therefore, refactoring may not be necessary in this method. Add comments to these methods describing possible adverse effects.

Long term, refactor the interface of `PlatformFile` to contain the potential race issues within the class. For example, move the `safePermissions` function into the `PlatformFile` class so that `hasSafePermissions` is not exposed outside of the class.

7. No limit on the amount of data that parsePlist can parse

Severity: **Low**	Difficulty: **Low**
Type: Denial of Service	Finding ID: TOB-ATL-7
Target: `osquery/utils/darwin/plist.mm`	

Description

To support several macOS-specific tables, osquery contains a function called `osquery::parsePlist`, which reads and parses *property list* (`.plist`) files by using the Apple Foundation framework class `NSPropertyListSerialization`. The `parsePlist` function is used by tables such as `browser_plugins` and `xprotect_reports` to read user-accessible files.

The function does not have any limit on the amount of data that it will parse.

```
id ns_path = [NSString stringWithUTF8String:path.string().c_str()];
id stream = [NSInputStream inputStreamWithFileAtPath:ns_path];
if (stream == nil) {
  return Status(1, "Unable to read plist: " + path.string());
}

// Read file content into a data object, containing potential plist data.
NSError* error = nil;
[stream open];
id plist_data = [NSPropertyListSerialization propertyListWithStream:stream
                                                            options:0
                                                             format:NULL
                                                              error:&error];
```

Figure 7.1: The `parsePlist` implementation does not have a limit on the amount of data that it can deserialize.

```
auto info_path = path + "/Contents/Info.plist";
// Ensure that what we're processing is actually a plug-in.
if (!pathExists(info_path)) {
  return;
}
if (osquery::parsePlist(info_path, tree).ok()) {
  // Plugin did not include an Info.plist, or it was invalid
  for (const auto& it : kBrowserPluginKeys) {
    r[it.second] = tree.get(it.first, "");

    // Convert bool-types to an integer.
    jsonBoolAsInt(r[it.second]);
  }
}
```

Figure 7.2: `browser_plugins` uses `parsePlist` on user-controlled files.

Exploit Scenario

An attacker crafts a large, valid `.plist` file and stores it in `~/Library/Internet Plug-Ins/foo/Contents/Info.plist` on a macOS system running the osquery daemon. When osquery executes a query using the `browser_plugins` table, it reads and parses the complete file, causing high resource consumption.

Recommendations

Short term, modify the `browser_plugins` and `xprotect_reports` tables to enforce a maximum file size (e.g., by combining `readFile` and `parsePlistContent`).

Long term, to prevent this issue in future tables, consider removing the `parsePlist` function or rewriting it as a helper function around a safer implementation.

8. The parsePlist function can hang on reading certain files

Severity: **Medium**	Difficulty: **Low**
Type: Denial of Service	Finding ID: TOB-ATL-8
Target: `osquery/utils/darwin/plist.mm`	

Description

The osquery codebase contains a function called `osquery::parsePlist`, which reads and parses `.plist` files. This function opens the target file directly by using the `inputStreamWithFileAtPath` method from `NSInputStream`, as shown in figure 7.1 in the previous finding, and passes the resulting input stream to `NSPropertyListSerialization` for direct consumption. However, `parsePlist` can hang on reading certain files. For example, if the file to be read is a FIFO, the function blocks the osquery thread until another program or thread opens the FIFO to write to it.

Exploit Scenario

An attacker creates a FIFO file on a macOS device in `~/Library/Internet Plug-Ins/foo/Contents/Info.plist` or `~/Library/Logs/DiagnosticReports/XProtect-foo` using the `mkfifo` command. The osquery agent attempts to open and read the file when building a response for queries on the `browser_plugins` and `xprotect_reports` tables, but `parsePlist` blocks the osquery thread indefinitely, leaving osquery unable to respond to the query request.

Recommendations

Short term, implement a check in `parsePlist` to verify that the `.plist` file to be read is not a special file.

Long term, introduce a timeout on file operations so that a block does not stall the osquery thread. Also consider replacing `parsePlist` in favor of the `parsePlistContent` function and standardizing all file reads on a single code path to prevent similar issues going forward.

9. The parseJSON function can hang on reading certain files on Linux and macOS

Severity: **Medium**	Difficulty: **Low**
Type: Denial of Service	Finding ID: TOB-ATL-9
Target: `osquery/filesystem/filesystem.cpp`	

Description

The osquery codebase contains a function called `osquery::parseJSON`, which reads and parses JSON files. This function opens the target file by passing a filename directly to `boost::property_tree::read_json`. On macOS and Linux, `parseJSON` can hang on reading certain files. For example, if the file to be read is a FIFO, the function blocks the osquery thread until another program or thread opens the FIFO to write to it. This function is currently used by the `firefox_addons` table.

```
Status parseJSON(const fs::path& path, pt::ptree& tree) {
  try {
    pt::read_json(path.string(), tree);
  } catch (const pt::json_parser::json_parser_error& /* e */) {
    return Status(1, "Could not parse JSON from file");
  }
  return Status::success();
}
```

Figure 9.1: osquery uses `boost::property_tree` to read and parse the file in the path.

```
void genFirefoxAddonsFromExtensions(const std::string& uid,
                                    const std::string& path,
                                    QueryData& results) {
  pt::ptree tree;
  if (!osquery::parseJSON(path + kFirefoxExtensionsFile, tree).ok()) {
    TLOG << "Could not parse JSON from: " << path + kFirefoxExtensionsFile;
    return;
  }
```

Figure 9.2: `parseJSON` reads the `extensions.json` file from the Firefox profile.

Exploit Scenario

An attacker creates a FIFO file named `extensions.json` using `mkfifo` and stores it on the Firefox profile path (e.g., in `~/.mozilla/firefox/bar/extensions.json` on a Linux system). The osquery agent attempts to respond to a query on the `firefox_addons` table and opens the file. `parseJSON` blocks the osquery thread indefinitely, leaving osquery unable to respond to further requests.

Recommendations

Short term, implement a check in `parseJSON` to verify that the JSON file to be read is not a special file.

Long term, introduce a timeout on file operations so that a block does not stall the osquery thread. Also consider replacing `osquery::parseJSON` in favor of the `parseJSONContent` function or the `osquery::JSON` helpers and standardizing all file reads on a single code path to prevent similar issues going forward.

10. No limit on the amount of data read or expanded from the Safari extensions table

Severity: **Low**	Difficulty: **Low**
Type: Denial of Service	Finding ID: TOB-ATL-10
Target: `osquery/tables/applications/darwin/browser_plugins.cpp`	

Description

The `safari_extensions` table allows the agent to query installed Safari extensions on a certain user profile. Said extensions consist of extensible archive format (XAR) compressed archives with the `.safariextz` file extension, which are stored in the `~/Library/Safari/Extensions` folder.

The osquery program does not have a limit on the amount of data that can be processed when reading and inflating the Safari extension contents; a large amount of data may cause a denial of service.

```
xar_t xar = xar_open(path.c_str(), READ);
if (xar == nullptr) {
  TLOG << "Cannot open extension archive: " << path;
  return;
}

xar_iter_t iter = xar_iter_new();
xar_file_t xfile = xar_file_first(xar, iter);

size_t max_files = 500;
for (size_t index = 0; index < max_files; ++index) {
  if (xfile == nullptr) {
    break;
  }

  char* xfile_path = xar_get_path(xfile);
  if (xfile_path == nullptr) {
    break;
  }

  // Clean up the allocated content ASAP.
  std::string entry_path(xfile_path);
  free(xfile_path);
  if (entry_path.find("Info.plist") != std::string::npos) {
    if (xar_verify(xar, xfile) != XAR_STREAM_OK) {
      TLOG << "Extension info extraction failed verification: " << path;
    }

    size_t size = 0;
    char* buffer = nullptr;
    if (xar_extract_tobuffersz(xar, xfile, &buffer, &size) != 0 ||
        size == 0) {
      break;
    }

    std::string content(buffer, size);
    free(buffer);

    pt::ptree tree;
    if (parsePlistContent(content, tree).ok()) {
      for (const auto& it : kSafariExtensionKeys) {
        r[it.second] = tree.get(it.first, "");
      }
    }
    break;
  }
  xfile = xar_file_next(iter);
}
```

Figure 10.1: `genSafariExtension` extracts the full `Info.plist` to memory.

Exploit Scenario

An attacker crafts a valid extension containing a large `Info.plist` file and stores it in `~/Library/Safari/Extensions/foo.safariextz`. When the osquery agent attempts to respond to a query on the `safari_extensions` table, it opens the archive and expands the full `Info.plist` file in memory, causing high resource consumption.

Recommendations

Short term, enforce a limit on the amount of information that can be extracted from an XAR archive.

Long term, add guidelines to the development documentation on handling untrusted input data. For instance, advise developers to limit the amount of data that may be ingested, processed, or read from untrusted sources such as user-writable files. Enforce said guidelines by performing code reviews on new contributions.

11. Extended attributes table may read uninitialized or out-of-bounds memory

Severity: **Medium**	Difficulty: **Medium**
Type: Timing	Finding ID: TOB-ATL-11
Target: `osquery/tables/system/darwin/extended_attributes.cpp`	

Description

The `extended_attributes` table calls the `listxattr` function twice: first to query the extended attribute list size and then to actually retrieve the list of attribute names. Additionally, the return values from the function calls are not checked for errors. This leads to a race condition in the `parseExtendedAttributeList` osquery function, in which the content buffer is left uninitialized if the target file is deleted in the time between the two `listxattr` calls. As a result, `std::string` will consume uninitialized and unbounded memory, potentially leading to out-of-bounds memory reads.

```
std::vector<std::string> attributes;
ssize_t value = listxattr(path.c_str(), nullptr, (size_t)0, 0);
char* content = (char*)malloc(value);
if (content == nullptr) {
  return attributes;
}

ssize_t ret = listxattr(path.c_str(), content, value, 0);
if (ret == 0) {
  free(content);
  return attributes;
}

char* stable = content;
do {
  attributes.push_back(std::string(content));
  content += attributes.back().size() + 1;
} while (content - stable < value);
free(stable);
return attributes;
```

Figure 11.1: `parseExtendedAttributeList` calls `listxattr` twice.

Exploit Scenario

An attacker creates and runs a program to race osquery while it is fetching extended attributes from the file system. The attacker is successful and causes the osquery agent to crash with a segmentation fault.

Recommendations

Short term, rewrite the affected code to check the return values for errors. Replace `listxattr` with `flistxattr`, which operates on opened file descriptors, allowing it to continue to query extended attributes even if the file is removed (`unlink`-ed) from the file system.

Long term, establish and enforce best practices for osquery contributions by leveraging automated tooling and code reviews to prevent similar issues from reoccurring. For example, use file descriptors instead of file paths when you need to perform more than one operation on a file to ensure that the file is not deleted or replaced mid-operation. Consider using static analysis tools such as CodeQL to look for other instances of similar issues in the code and to detect new instances of the problem on new contributions.

12. The readFile function can hang on reading a file

Severity: **Medium**	Difficulty: **Low**
Type: Denial of Service	Finding ID: TOB-ATL-12
Target: `osquery/filesystem/filesystem.cpp`	

Description

The `readFile` function is used to provide a standardized way to read files. It uses the `read_max` variable to prevent the functions from reading excessive amounts of data. It also selects one of two modes, blocking or non-blocking, depending on the file properties. When using the blocking approach, it reads `block_size`-sized chunks of the file, with a minimum of 4096 bytes, and returns the chunks to the caller. However, the call to `read` can block the osquery thread when reading certain files.

```
if (handle.blocking_io) {
  // Reset block size to a sane minimum.
  block_size = (block_size < 4096) ? 4096 : block_size;
  ssize_t part_bytes = 0;
  bool overflow = false;
  do {
    std::string part(block_size, '\0');
    part_bytes = handle.fd->read(&part[0], block_size);
    if (part_bytes > 0) {
      total_bytes += static_cast<off_t>(part_bytes);
      if (total_bytes >= read_max) {
        return Status::failure("File exceeds read limits");
      }
      if (file_size > 0 && total_bytes > file_size) {
        overflow = true;
        part_bytes -= (total_bytes - file_size);
      }
      predicate(part, part_bytes);
    }
  } while (part_bytes > 0 && !overflow);
} else {
```

Figure 12.1: The `blocking_io` flow can stall.

Exploit Scenario

An attacker creates a symlink to `/dev/tty` in a path known to be read by the osquery agent. When the osquery agent attempts to read the file, it stalls.

Recommendations

Short term, ensure that the file operations in `filesystem.cpp` do not block the osquery thread.

Long term, introduce a timeout on file operations so that a block does not stall the osquery thread.

13. The POSIX PlatformFile constructor may block the osquery thread	
Severity: **Medium**	Difficulty: **Low**
Type: Denial of Service	Finding ID: TOB-ATL-13
Target: `osquery/filesystem/posix/fileops.cpp`	

Description

The POSIX implementation of `PlatformFile`'s constructor uses the open syscall to obtain a handle to the file. If the file to be opened is a FIFO, the call to open blocks the osquery thread unless the `O_NONBLOCK` flag is used. There are several places in the codebase in which the constructor is called without the `PF_NONBLOCK` flag set; all of these calls may stall on opening a FIFO.

```
PlatformFile::PlatformFile(const fs::path& path, int mode, int perms)
    : fname_(path) {
  ...
  if ((mode & PF_NONBLOCK) == PF_NONBLOCK) {
    oflag |= O_NONBLOCK;
    is_nonblock_ = true;
  }

  if ((mode & PF_APPEND) == PF_APPEND) {
    oflag |= O_APPEND;
  }

  if (perms == -1 && may_create) {
    perms = 0666;
  }

  boost::system::error_code ec;
  if (check_existence &&
      (!fs::exists(fname_, ec) || ec.value() != errc::success)) {
    handle_ = kInvalidHandle;
  } else {
    handle_ = ::open(fname_.c_str(), oflag, perms);
  }
}
```

Figure 13.1: The POSIX `PlatformFile` constructor

```
./filesystem/file_compression.cpp:26:  PlatformFile inFile(in, PF_OPEN_EXISTING | PF_READ);
./filesystem/file_compression.cpp:32:  PlatformFile outFile(out, PF_CREATE_ALWAYS | PF_WRITE);
./filesystem/file_compression.cpp:102:  PlatformFile inFile(in, PF_OPEN_EXISTING | PF_READ);
./filesystem/file_compression.cpp:108:  PlatformFile outFile(out, PF_CREATE_ALWAYS | PF_WRITE);
./filesystem/file_compression.cpp:177:    PlatformFile pFile(f, PF_OPEN_EXISTING | PF_READ);
./filesystem/filesystem.cpp:242:    PlatformFile fd(path, PF_OPEN_EXISTING | PF_WRITE);
./filesystem/filesystem.cpp:258:    PlatformFile fd(path, PF_OPEN_EXISTING | PF_READ);
./filesystem/filesystem.cpp:531:  PlatformFile fd(path, PF_OPEN_EXISTING | PF_READ);
./carver/carver.cpp:230:    PlatformFile src(srcPath, PF_OPEN_EXISTING | PF_READ);
```

Figure 13.2: Uses of `PlatformFile` without PF_NONBLOCK

Exploit Scenario

An attacker creates a FIFO file that is opened by one of the functions above, stalling the osquery agent.

Recommendations

Short term, investigate the uses of `PlatformFile` to identify possible blocks.

Long term, use a static analysis tool such as CodeQL to scan the code for instances in which uses of the `open` syscall block the osquery thread.

14. No limit on the amount of data the Carver::blockwiseCopy method can write

Severity: **Medium**	Difficulty: **Low**
Type: Denial of Service	Finding ID: TOB-ATL-14
Target: `osquery/carver/carver.cpp`	

Description

The `Carver::blockwiseCopy` method copies files matching a requested carving operation. It does so by reading a block of `carver_block_size` bytes from the source file and then writing the data to the destination file. However, the copy loop does not check `read_max`, so there is no limit on how much data can be written. Furthermore, the source file is not opened using the `PF_NONBLOCK` flag, so the reading operation can block the osquery thread. Refer to TOB-ATL-12 for more detail.

```
Status Carver::blockwiseCopy(PlatformFile& src, PlatformFile& dst) {
  auto blkCount = ceil(static_cast<double>(src.size()) /
                       static_cast<double>(FLAGS_carver_block_size));

  std::vector<char> inBuff(FLAGS_carver_block_size, 0);
  for (size_t i = 0; i < blkCount; i++) {
    auto bytesRead = src.read(inBuff.data(), FLAGS_carver_block_size);
    if (bytesRead > 0) {
      auto bytesWritten = dst.write(inBuff.data(), bytesRead);
      if (bytesWritten < 0) {
        return Status(1, "Error writing bytes to tmp fs");
      }
    }
  }

  return Status::success();
};
```

Figure 14.1: The `Carver::blockwiseCopy` method has no limit on the amount of data it can write.

Exploit Scenario

An attacker creates a large sparse file on disk in a location in which a carving operation is performed. When the carving operation is performed, the large file consumes the available space on the destination file system.

Recommendations

Short term, implement `read_max` in `blockwiseCopy` to limit the size of the data that it can copy.

Long term, refactor the code to eliminate the need to copy files. For example, refactor the code to read the files in place when adding them to the archive or create symlinks in the target directory.

15. The carves table truncates large file sizes to 32 bits

Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-ATL-15
Target: `osquery/tables/forensic/carves.cpp`	

Description

The `enumerateCarves` function uses `rapidjson::Value::GetInt()` to retrieve a size value from a JSON string. The `GetInt` return type is `int`, so it cannot represent sizes exceeding 32 bits; as a result, the size of larger files will be truncated.

```
void enumerateCarves(QueryData& results, const std::string& new_guid) {
  std::vector<std::string> carves;
  scanDatabaseKeys(kCarves, carves, kCarverDBPrefix);

  for (const auto& carveGuid : carves) {
    std::string carve;
    auto s = getDatabaseValue(kCarves, carveGuid, carve);
    if (!s.ok()) {
      VLOG(1) << "Failed to retrieve carve GUID";
      continue;
    }

    JSON tree;
    s = tree.fromString(carve);
    if (!s.ok() || !tree.doc().IsObject()) {
      VLOG(1) << "Failed to parse carve entries: " << s.getMessage();
      return;
    }

    Row r;
    if (tree.doc().HasMember("time")) {
      r["time"] = INTEGER(tree.doc()["time"].GetUint64());
    }

    if (tree.doc().HasMember("size")) {
      r["size"] = INTEGER(tree.doc()["size"].GetInt());
    }

    stringToRow("sha256", r, tree);
    stringToRow("carve_guid", r, tree);
    stringToRow("request_id", r, tree);
    stringToRow("status", r, tree);
    stringToRow("path", r, tree);

    // This table can be used to request a new carve.
    // If this is the case then return this single result.
    auto new_request = (!new_guid.empty() && new_guid == r["carve_guid"]);
    r["carve"] = INTEGER((new_request) ? 1 : 0);

    results.push_back(r);
  }
}
} // namespace
```

Figure 15.1: The `enumerateCarves` function truncates files of large sizes.

Exploit Scenario

An attacker creates a file on disk of a size that overflows 32 bits by only a small amount, such as `0x100001336`. The `carves` tables reports the file size incorrectly as `0x1336` bytes. The attacker bypasses checks based on the reported file size.

Recommendations

Short term, use `GetUint64` instead of `GetInt` to retrieve the file size.

Long term, use static analysis tools such as CodeQL to look for other instances in which a type of size `int` is retrieved from `JSON` and stored in an `INTEGER` field.

16. The time table may not null-terminate strings correctly

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-ATL-16
Target: `osquery/tables/utility/time.cpp`	

Description

The osquery `time` table uses `strftime` to format time information, such as the time zone name, into user-friendly strings. If the amount of information to be written does not fit in the provided buffer, `strftime` returns zero and leaves the buffer contents in an undefined state.

```
QueryData genTime(QueryContext& context) {
  Row r;

  time_t osquery_time = getUnixTime();

  struct tm gmt;
  gmtime_r(&osquery_time, &gmt);
  struct tm now = gmt;
  auto osquery_timestamp = toAsciiTime(&now);

  char local_timezone[5] = {0};
  {
    struct tm local;
    localtime_r(&osquery_time, &local);
    strftime(local_timezone, sizeof(local_timezone), "%Z", &local);
  }

  char weekday[10] = {0};
  strftime(weekday, sizeof(weekday), "%A", &now);

  char timezone[5] = {0};
  strftime(timezone, sizeof(timezone), "%Z", &now);
```

Figure 16.1: `genTime` uses `strftime` to get the time zone name and day of the week.

The strings to be written vary depending on the locale configuration, so some strings may not fit in the provided buffer. The code does not check the return value of `strftime` and assumes that the string buffer is always null-terminated, which may not always be the case.

Exploit Scenario

An attacker finds a way to change the time zone on a system in which `%Z` shows the full time zone name. When the osquery agent attempts to respond to a query on the `time` table, it triggers undefined behavior.

Recommendations

Short term, add a check to verify the return value of each `strftime` call made by the table implementation. If the function returns zero, ensure that the system writes a valid string in the buffer before it is used as part of the table response.

Long term, perform code reviews on new contributions and consider using automated code analysis tools to prevent these kinds of issues from reoccurring.

17. The elf_info table can crash the osquery agent

Severity: **Medium**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-ATL-17
Target: `osquery/tables/system/linux/elf_info.cpp`	

Description

The `elf_info` table uses the `libelfin` library to read properties of ELF files. The library maps the entire ELF binary into virtual memory and then uses memory accesses to read the data. Specifically, the `load` method of the `mmap_loader` class returns a pointer to the data for a given offset and size. To ensure that the memory access stays within the bounds of the memory-mapped file, the library checks that the result of adding the offset and the size is less than the size of the file. However, this check does not account for the possibility of overflows in the addition operation. For example, an offset of `0xffffffffffffffff` and a size of 1 would overflow to the value `0`. This makes it possible to bypass the check and to create references to memory outside of the bounds. The `elf_info` table indirectly uses this function when loading section headers from an ELF binary.

```
class mmap_loader : public loader
{
        void *base;
        size_t lim;

public:
        mmap_loader(int fd)
        {
                off_t end = lseek(fd, 0, SEEK_END);
                if (end == (off_t)-1)
                        throw system_error(errno, system_category(),
                                           "finding file length");
                lim = end;

                base = mmap(nullptr, lim, PROT_READ, MAP_SHARED, fd, 0);
                if (base == MAP_FAILED)
                        throw system_error(errno, system_category(),
                                           "mmap'ing file");
                close(fd);
        }
        ...
        const void *load(off_t offset, size_t size)
        {
                if (offset + size > lim)
                        throw range_error("offset exceeds file size");
                return (const char*)base + offset;
        }
};
```

Figure 17.1: The `libenfin` *library's limit check does not account for overflows.*

Exploit Scenario

An attacker knows of a writable path in which osquery scans ELF binaries. He creates a malformed ELF binary, causing the pointer returned by the vulnerable function to point to an arbitrary location. He uses this to make the osquery agent crash, leak information from the process memory, or circumvent address space layout randomization (ASLR).

Recommendations

Short term, work with the developers of the `libelfbin` project to account for overflows in the check.

Long term, implement the recommendations in TOB-ATL-2 to minimize the impact of similar issues.

Summary of Recommendations

The open-source osquery project is a work in progress with multiple planned iterations. Trail of Bits recommends that the findings detailed in this report be addressed and that the following additional steps be taken:

- Use techniques such as sandboxing, and/or separate operations into intercommunicating processes with different security contexts to ensure that untrusted data from the system is parsed in a security context with the lowest privilege level possible.

- As a stopgap measure before implementing such extensive restructuring efforts, consider adding additional exploit mitigations to the osquery agent, drawing on the latest developments in the toolchains. For example, the Clang compiler can use SafeStack and code flow integrity (CFI) protections. However, enabling Clang CFI on the osquery project may be difficult, possibly requiring considerable time and effort because of the large number of dependencies and the fact that it needs to use LTO. See Appendix C.

- Expand the watchdog capabilities to include a heartbeat functionality that can detect when the process stalls without excessive resource use, such as blocking syscalls.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories

Category	Description
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories

Category	Description
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Configuration	The configuration of system components in accordance with best practices
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Data Handling	The safe handling of user inputs and data processed by the system
Documentation	The presence of comprehensive and readable codebase documentation
Maintenance	The timely maintenance of system components to mitigate risk
Memory Safety and Error Handling	The presence of memory safety and robust error-handling mechanisms
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Compiler Mitigations

NX makes the data sections (including the stack and heap) of the program non-executable. This makes it more difficult for an attacker to execute shellcode. Attackers normally use return-oriented programming (ROP) to bypass NX. Forcing attackers to use ROP makes exploits less reliable across different builds of a program. This mitigation is enabled by default.

Stack canaries (also known as stack cookies) make buffer overflow vulnerabilities more difficult to exploit. A stack canary is a global, randomly generated value that is copied to the stack between the stack variables and stack metadata in a function's prologue. When a function returns, the canary on the stack is checked against the global value. The program exits if there is a mismatch. This makes it more difficult for an exploit to overwrite the return address on the stack. Depending on the circumstances, attackers may bypass this mitigation by leaking the canary with a separate information leak vulnerability or by brute-forcing the canary byte-by-byte.

ASLR randomizes where each section of the program is placed in memory. This makes it more difficult for an attacker to write reliable exploits, primarily by making it more difficult to jump to ROP gadgets. ASLR requires cooperation from both the system and the compiler. In order to support ASLR fully, a program must be compiled as a position-independent executable (PIE). Most of the Linux distributions have ASLR enabled. This can be checked by reading the value stored in the `/proc/sys/kernel/randomize_va_space` file: 0 means that ASLR is disabled, 1 means it is partially enabled (fewer bits of the addresses are randomized), and 2 means it is fully enabled. This file is writable, and an admin can disable or enable this mitigation. ASLR may be bypassed if an attacker has an information leak in the program.

Relocations read-only (RELRO) is a mitigation technique to harden the data sections of an ELF process. It has three modes of operation: disabled, partial, and full. When a program uses a function from a dynamically loaded library, this function address is stored in the Global Offset Table for the Procedure Linkage Table (`.got.plt`) section. When RELRO is disabled, the function addresses in `.got.plt` point to a dynamic resolver function that resolves the given function address when it is called for the first time. In this case, the memory in which the address is stored is both readable and writable. Because of that, an attacker who has control over the process control flow can change the entry of a given function in `.got.plt` to point to any other executable address. For example, the attacker can change the `puts` function's `.got.plt` entry to point to a `system` function. Then, if the program calls `puts(“bin/sh”)`, a `system(“/bin/sh”)` would be called instead. When RELRO is fully enabled, the dynamic resolver resolves all of the addresses on program startup and changes the permissions of data sections (and, therefore, `.got.plt`) to read-only.

`FORTIFY_SOURCE` is a `glibc`-specific feature that enables a series of mitigations primarily aimed at preventing buffer overflows. With a `FORTIFY_SOURCE` level of 1, `glibc` will add compile-time warnings when potentially unsafe calls to common `libc` functions (e.g., `memcpy` and `strcpy`) are made. With a `FORTIFY_SOURCE` level of 2, `glibc` will add more stringent run-time checks to these functions. Additionally, `glibc` will enable a number of lesser-known mitigations. For example, it will disallow the use of a `%n` format specifier in format strings that are not located in read-only memory pages. This prevents attackers from overwriting data (and gaining code execution) with format string vulnerabilities.

Stack clash protection mitigates a "stack clash vulnerability," in which a program's stack memory region grows so much that it overlaps with another memory region. This bug makes the program confuse two different memory addresses (stack and heap, for example) so that some of their data overlaps, leading to a denial of service or to control flow hijacking. The stack clash protection mitigation works by adding explicit memory probing to functions that allocate a lot of stack memory so that the function's stack allocation will never make the stack pointer jump over the stack memory guard page, which is located before the stack.

CFI mitigates various control-flow hijack attempts and makes it harder to exploit vulnerabilities such as use-after-free or to use exploitation techniques such as ROP. This mitigation is currently implemented only in Clang/LLVM, and in Visual Studio. (Clang 13: Control Flow Integrity)

SafeStack makes it harder to exploit stack-based buffer overflows, as it separates the program stack into the safe stack (which holds saved registers and return addresses) and the unsafe stack (which stores everything else). This mitigation is currently implemented only in Clang/LLVM. (Clang 13: SafeStack)

D. CVEs in Dependencies

The following table lists the majority of the dependencies used by osquery. For each dependency, the version used by osquery and a non-exhaustive list of recently reported CVEs affecting the dependency are indicated. CVEs that are fixed in a more recent version than the one used by osquery are highlighted in red. Note that this does not mean that osquery is impacted by said CVE, since osquery may not build the affected component or execute the problematic code path. Rather, this table is meant to highlight the current lack of a process to detect such vulnerabilities.

We have reviewed the potentially problematic (red) CVEs in this table and found that only the problems in `libelfin` and `expat` are relevant; the other issues in the other CVEs do not adversely affect osquery. Regarding the vulnerabilities in the other dependencies, osquery does not execute the effected code (e.g., in `libgcrypt`, `librpm`, `lldpd`, `zstd`) or uses only header file definitions from it (e.g., `libaudit`).

Recent CVEs Affecting osquery Dependencies

Dependency	Version and Release Date	Recent CVEs
`augeas`	1.12.0 (d133d97), 2019-04-13	CVE-2017-7555
`boost`	1.77.0 (boost-1.77.0), 2021-08-04	-
`bzip2`	1.0.8 (bzip2-1.0.8), 2019-07-13	CVE-2019-12900
`dbus`	1.12.20 (dbus-1.12.20), 2020-07-02	CVE-2020-35512, CVE-2020-12049, CVE-2019-12749
`ebpfpub`	a2458c3, 2021-10-02	-
`expat`	2.2.10 (R_2_2_10), 2020-10-03	CVE-2022-22827, CVE-2022-22826, CVE-2022-22825, CVE-2022-22824, CVE-2022-22823, CVE-2022-22822, CVE-2021-46143, CVE-2021-45960, CVE-2019-15903, CVE-2018-20843, CVE-2017-9233, CVE-2013-0340
`gflags`	2.2.2 (v2.2.2), 2018-11-11	-
`glog`	0.5.0 (v0.5.0), 2021-05-07	-
`gnulib`	91584ed, 2019-04-07	CVE-2018-17942, CVE-2017-7476
`googletest`	1.11.0 (release-1.11.0), 2021-06-12	-

libarchive	3.5.2 (v3.5.2), 2021-08-22	CVE-2021-36976, CVE-2020-21674, CVE-2020-9308, CVE-2019-1000020, CVE-2019-1000019, CVE-2019-19221, CVE-2019-11463, CVE-2018-1000880, CVE-2018-1000879, CVE-2018-1000878, CVE-2018-1000877
libaudit	2.4.2 (`20204cc`), 2015-04-28	CVE-2015-5186
libcap	1.2.59 (v1.2.59), 2021-09-26	-
libcryptsetup	1.7.5 (v1_7_5), 2017-04-27	CVE-2021-4122, CVE-2020-14382
libdevmapper	2.02.173 (v2_02_173), 2017-07-20	-
libdpkg	1.19.0.5 (`a059dd6`), 2018-01-17	-
libelfin	0.3 (v0.3), 2016-12-03	CVE-2020-24827, CVE-2020-24826, CVE-2020-24825, CVE-2020-24824, CVE-2020-24823, CVE-2020-24822, CVE-2020-24821
libgcrypt	1.8.1 (`libgcrypt-1.8.1`), 2017-08-27	CVE-2021-40528, CVE-2021-33560, CVE-2021-3345, CVE-2019-12904, CVE-2018-6829, CVE-2018-0495, CVE-2017-9526, CVE-2017-7526, CVE-2017-0379
libgpg-error	1.27 (`libgpg-error-1.27`), 2017-02-28	-
libiptables	1.8.3 (v1.8.3), 2019-05-27	CVE-2019-11360
libmagic	5.40 (FILE5_40), 2021-03-30	-
librdkafka	1.8.0-RC3 (v1.8.0-RC3), 2021-09-15	-
librpm	4.16.1.2 (`278883a`), 2020-12-16	CVE-2021-35939, CVE-2021-35938, CVE-2021-35937, CVE-2021-20271, CVE-2021-20266, CVE-2021-3521, CVE-2021-3421
libsmartctl	0.3.1 (`f8b3b6d`), 2018-05-17	-
libudev	174 (174), 2011-10-19	-
libxml2	2.9.12 (v2.9.12), 2021-05-13	CVE-2021-3541, CVE-2021-3537, CVE-2021-3518, CVE-2021-3517, CVE-2020-24977, CVE-2020-7595,

		CVE-2019-20388, CVE-2019-19956, CVE-2018-14567, CVE-2018-14404, CVE-2018-9251
linenoise-ng	4754bee, 2017-06-28	-
lldpd	0.9.6 (0.9.6), 2017-01-21	CVE-2021-43612, CVE-2020-27827
lzma	5.2.5 (v5.2.5), 2020-03-17	-
openssl	1.1.1l, 2021-08-24	CVE-2021-3712, CVE-2021-3711, CVE-2021-3450, CVE-2021-3449, CVE-2021-23841, CVE-2021-23840, CVE-2020-1971, CVE-2020-1967, CVE-2019-1551, CVE-2019-1563, CVE-2019-1549, CVE-2019-1547, CVE-2019-1552, CVE-2019-1543, CVE-2018-0734, CVE-2018-0735
popt	1.16 (1.16), 2017-06-14	-
rapidjson	1a825d24, 2019-10-14	-
rocksdb	6.22.1 (51b5409), 2021-06-25	-
sleuthkit	4.11.0 (sleuthkit-4.11.0), 2021-08-02	CVE-2020-10233, CVE-2020-10232, CVE-2019-1010065, CVE-2019-14532, CVE-2019-14531, CVE-2018-19497, CVE-2018-11740, CVE-2018-11739, CVE-2018-11738, CVE-2018-11737
sqlite	3.37.0 (3.37.0-1), 2021-11-27	CVE-2021-20227, CVE-2020-15358, CVE-2020-13871, CVE-2020-13632, CVE-2020-13631, CVE-2020-13630, CVE-2020-13435, CVE-2020-13434, CVE-2020-11656, CVE-2020-11655, CVE-2020-9327, CVE-2020-6405, CVE-2019-20218, CVE-2019-19959, CVE-2019-19926, CVE-2019-19925, CVE-2019-19924, CVE-2019-19923, CVE-2019-19317
ssdeep-cpp	d8705da, 2019-02-21	-
thrift	0.15.0 (v0.15.0), 2021-09-04	CVE-2020-13949, CVE-2019-0210, CVE-2019-0205, CVE-2018-11798, CVE-2018-1320

`util-linux`	2.27.1 (v2.27.1), 2015-11-02	CVE-2021-37600, CVE-2018-7738, CVE-2017-2616, CVE-2016-5011, CVE-2016-2779
`yara`	4.1.3 (v4.1.3), 2021-10-21	CVE-2021-3402, CVE-2019-19648, CVE-2019-5020, CVE-2018-19976, CVE-2018-19975, CVE-2018-19974, CVE-2018-12035, CVE-2018-12034
`zlib`	1.2.11 (v1.2.11), 2017-01-15	-
`zstd`	1.4.0 (v1.4.0), 2019-04-16	CVE-2021-24032, CVE-2021-24031, CVE-2019-11922

E. Code Quality Recommendations

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of bugs and vulnerabilities in the future.

- Update the use of `readlink` in `proc.cpp` to account for truncated paths.
- Reduce the use of "magic" strings and numbers and replace them with constants. Magic strings and numbers make code harder to read and can result in hard-to-spot problems. For example, there is a mistyped magic string in `Registry::call` (*"detatch"*), which causes a problem with table deregistration in Automated Table Construction (ATC). This issue could have been prevented if constants or definitions were used for said value.
- Review the accuracy of comments when code is refactored or changed. For example, the comment on `carver_block_size` indicates that it is used for POSTing data, but it has other uses as well. For example, `carver_block_size` is the block size for `blockwiseCopy`.
- Add a check to `blockwiseCopy` and `postCarve` in `carver.cpp` to ensure that a `carver_block_size` of zero does not cause a division-by-zero error.
- Change `sqlite3_result_int` to `sqlite3_result_int64` in `dynamic_table_row.cpp` so that integer values are not truncated to 32 bits.