OSQuery Application Security Assessment

Facebook

October 23, 2015 – Version 1.3

Prepared for
Mike Arpaia

Prepared by
Raphael Salas
Andrew Rahimi
Robert Seacord

Document Change Log		
Version	Date	Change
1.0	2015-10-23	Distributed report to Facebook.
1.1	2015-11-05	Incorporated discussions during readout regarding exploitability of key findings.
1.2	2016-01-18	Preparing for public release after rounds of additional peer review.
1.3	2016-01-25	Public release.

Executive Summary

Synopsis

Facebook engaged NCC Group to conduct a security assessment of the osquery instrumentation framework. This assessment was conducted by 3 consultants during the weeks of October 12th and October 19th, 2015. The entire application was in scope, as sourced from the project's official GitHub page. The Facebook team requested that formal status meetings be foregone in exchange for active collaboration between Facebook and NCC Group via email and GitHub. This resulted in several issues being fixed in real-time alongside the disclosure process.

Scope

NCC Group's assessment included osquery core and plugin interfaces, focusing on the following main attack surfaces:

- **Data Collection**: An analysis of the set of osquery tables used to represent operating system state information, including the plugin interface allowing table implementation via the Thrift API.
- **Kernel Integration**: A review of OS X specific functionality exposed to the `osqueryd` daemon through the included kernel extension.
- **Transport Security**: A review of the `osqueryd` plugin interface that exposes its subsystems remotely over authenticated TLS endpoints.
- **Database Security**: A review of the RocksDB instance used to house and cache information about the system.
- **Configuration Parsing & Secure Defaults**: Code review of plugins that read and parse configuration from disk, command line, or remote servers.

Testing was performed on the Git revision 4d0cd46 dated 10/09/2015.

Key Findings

The assessment discovered the following key findings:

- A Time-of-Check Time-of-Use TOCTOU issue in the `osquery::readFile` function used throughout the framework allowing an attacker to cause osquery to read arbitrary files where the attacker may not have sufficient permissions to do so.
- A parsing issue where the configuration parser expects a "query" JSON key resulting in `osqueryd` aborting due to an unhandled exception. This can be triggered remotely via the TLS configuration plugin to cause denial of service, provided an attacker gains access to an authenticated TLS endpoint.
- Comparisons between integers of different signs and unsigned integer wrap-around throughout the osquery code.

Non-security findings are listed in Appendix F.

Limitations

While NCC Group was able to review the majority of the targeted components, the assessment of individual tables was inconsistent. Table implementations are specific to the various operating systems supported by osquery making complete coverage across all supported platforms impractical in the time provided. Due to time constraints, NCC Group prioritized the assessment of common table functionality across supported platforms, with most testing performed on the Ubuntu 14.04 LTS Vagrant VM and OS X 10.10.3 (Yosemite) configurations. Consequently, details specific to other Linux distributions such as CentOS and FreeBSD table implementations only received a cursory review.

Strategic Recommendations

Consider implementing a least-privilege model, where privileges are temporarily dropped before performing I/O operations on behalf of an unprivileged user. For example, user-owned files should be accessed with the effective privileges of the user to avoid allowing unprivileged users to operate on restricted files.

Enforce coding guidelines and fix violations. Do not disable warnings such as differing signedness comparisons and continue to test sanitized versions, adding additional sanitizers. Instrumented executables should be tested more aggressively to identify potential issues at runtime.

Extend read limitations across tables, particularly those that are OS specific. For instance, size checking against read limitations in the Safari Extensions table can prevent decompression bombs that can negatively impact the result of a query. Consistently applying limitations can prevent this and other conditions where unprivileged users can interfere with normal operation.

Maintain and test an updated build chain. NCC Group was able to leverage the updated Clang 3.7 compiler to identify numerous defects that had not been previously detected by Clang 3.5. Facebook should also ensure that the latest versions of tools and software components, such as Thrift, are used in the product.

Project Dashboard

Target Metadata

Name	osquery
Type	Daemon
Language	C++
Environment	Testing (Vagrant-provisioned)

Engagement Data

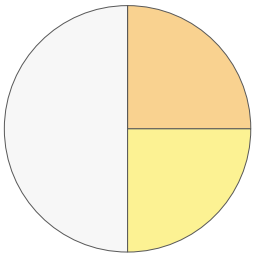
Type	Application Security Assessment
Method	Code-assisted
Dates	2015-10-12 to 2015-10-23
Consultants	3
Level of effort	6 person-weeks

Targets

osquery Source Code https://github.com/facebook/osquery – Reference commit: 4d0cd46

Vulnerability Breakdown

Critical Risk issues	0
High Risk issues	0
Medium Risk issues	2
Low Risk issues	2
Informational issues	4
Total issues	8



Category Breakdown

Data Exposure	1
Data Validation	5
Denial of Service	2

Key

Critical High Medium Low Informational

Synopsis

The entire osquery codebase is open source, however the nature of the application requires that it read and act on system-specific data that is inherently non-uniform across different systems and platforms. Therefore, the NCC Group team divided resources between targeted source code review, dynamic testing of the application, and minimal fuzzing.

Throughout the assessment, the Facebook team was extremely helpful and responsive in eliminating roadblocks and providing insight into the security considerations and architecture of the product. This was particularly beneficial in the initial stages of the assessment, allowing for a strong start. The Facebook team provided extensive documentation,[1] which allowed the NCC Group team to easily enumerate application functionality and identify attack surfaces. The Facebook team was also extremely responsive to issues identified via their GitHub or email and worked with the NCC Group team to patch vulnerabilities in real time.

From NCC Group, the testing team was led by Raphael Salas, with additional testing performed by Andrew Rahimi and Robert Seacord. The project manager was Deanna Bjorkquist. The account manager for Facebook is Bryan Solari.

From Facebook, the project was managed by Mike Arpaia, with technical support provided by Ted Reed. Peter Oehlert is the product security director.

Test Plan

NCC Group installed the Clang 3.7 compiler and performed various tests using the `-fsanitize` compiler flag. The team also used `valgrind`, a freely available tool.

NCC Group tested the transport security, filesystem security, permissions, and default configurations of osquery on multiple platforms to ensure wide coverage across supported OS configurations.

A test extension based on the `osquery/examples/example_extension.cpp` example was constructed to verify the rights of extensions through osquery's Thrift API. Extensions in osquery operate on whitelist principles. NCC Group verified that safe file permissions are enforced on directories that contain extension files when the extension autoload feature is in use. Like any extension framework, extension functionality in osquery can be abused by malicious extensions. This risk is accepted as osquery provides options for secure configuration deployment and management.

osquery mandates TLS for all HTTP socket connections including Configuration, Logger, Enrollment, and Distributed Queries. The application was evaluated exclusively as a TLS/HTTP client, and was confirmed to support only sufficiently strong, modern cipher suites. A testing HTTP server was provided in `osquery/tools/tests/test_http_server.py` that was used in conjunction with dummy certificates and TLS man-in-the-middle attempts to validate `osqueryd` functionality with each of the TLS-enabled features. Note that the security of the included testing HTTP server itself was not evaluated.

The NCC Group team performed light fuzzing of various osquery file inputs used by tables and configurations. For the majority of fuzz tests, the tool afl-fuzz[2] was used.

For the OS X kernel extension, the team built the kernel extension for debugging and compiled a simple program based on existing code throughout osquery to interact with the device node.

[1] https://osquery.readthedocs.org/
[2] http://lcamtuf.coredump.cx/afl/

Maintain and test an updated build chain

Maintaining an up to date tool environment is time consuming, but important. Tools, particularly compilers and their libraries, frequently add new static analysis features, new optimizations, new runtime checks, and improvements in the runtime system libraries.

The use of newer compilers, even on a limited number of platforms, has advantages for the entire code base. Newer static analysis features can detect problems that exist on all platforms, but are only reported by a newer compiler versions available on some of the platforms. Similarly, improvements in runtime analysis (e.g., `AddressSanitizer`) can find additional defects in all versions of the code. Improvements to the runtime libraries will only benefit newer builds that use the newer runtime. In general, developers should restrict themselves to language features that are common to all versions of the compilers being used, although it should also be possible to conditionally use newer compiler features when available.

The current build of osquery on Ubuntu 14.01 uses Clang 3.5. Clang 3.6 is available from official repositories, Clang 3.7 can be downloaded and manually extracted, and post 3.7 versions can be built from scratch. Upgrading these tools as part of the NCC Group assessment helped identify numerous defects that had not been previously detected. Consequently, we recommend upgrading to newer tool versions when available and feasible.

Upgrading compiler versions always has a non-zero cost, and consequently is never an automatic decision. In certain cases, Continuous Integration can also introduce obstacles for updating to the latest tool versions. New compiler versions can break existing code, for example, by taking advantage of undefined behavior in the code to provide new code optimizations. These impacts can be minimized by finding and eliminating undefined behavior in the code base.

Maintain and test up-to-date third-party components

As osquery relies on a number of third-party components, NCC Group generally recommends that Facebook adopt the latest versions of components. Defects are continually being discovered and repaired in these subsystems. NCC Group testing, for example, discovered a memory leak and other defects in Thrift. The osquery build currently uses Thrift 0.9.1 while the latest version is Thrift 0.9.3, which has repaired these defects.

Continually updating component versions used by osquery ensures that osquery incorporates the latest vulnerability patches and bug fixes.

Enforce coding guidelines and fix violations

NCC Group identified several informational issues, such as finding NCC-FBOSQ-003 on page 14 and finding NCC-FBOSQ-002 on page 15, that can cause errors, crashes, and potentially introduce security vulnerabilities; although no exploitable vulnerabilities were directly found from these issues. Enabling the relevant compiler flags to emit warnings and running tests with instrumented builds is encouraged to reveal potential issues and facilitate adherence to coding guidelines. NCC Group recommends reviewing instances in the code where integer wrap-around and integer comparison across different signs occur, and eliminating these behaviors.

Consider implementing a least-privilege model

The time-of-check, time-of-use (TOCTOU) race condition described in finding NCC-FBOSQ-006 on page 10 stems from a general approach that attempts to prevent various scenarios involving symbolic links and user FIFO files.

In general, checking for the existence of symbolic links is not the correct way to secure I/O operations. Symbolic links are a normal part of the operating environment and should be supported by the application. Additionally, checking for symbolic links does not solve similar problems such as hard links or accessing device files and other special files.

The correct way to handle these situations is to use operating system privilege management features. In no case should an unprivileged user be allowed to specify the name of a file that a privileged process will operate on. Privileged processes should never operate on user-owned files in user directories or in any temporary directories. If a privileged process must operate on a user-supplied file or in a user directory, the privileged process must first (temporarily) drop effective privileges to those of the user. After dropping privileges, it is safe to operate in user directories. It is also safe

to operate on symbolic links and hard links because, without elevated privileges, `osqueryd` will not be able to operate on any files for which the user lacks appropriate privileges.

Table of Vulnerabilities

For each finding, NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. For an explanation of NCC Group's risk rating and vulnerability categorization, see Appendix A on page 18.

Title	ID	Risk
Race Condition Allows Arbitrary File Read	006	Medium
Safari Extensions Table Allows Uncontrolled Resource Consumption	008	Medium
Read Limits Not Enforced for Property Lists	007	Low
Uncaught Exception in JSON Parsing Causes Abort	005	Low
Comparison of Integers of Different Signs	003	Informational
Unsigned Integer Wrap-around	002	Informational
Undefined Behaviors	004	Informational
Memory Leak in Core Application	001	Informational

Vulnerability Details

Vulnerability **Race Condition Allows Arbitrary File Read**

Risk **Medium** Impact: High, Exploitability: Low

Identifier NCC-FBOSQ-006

Category Data Exposure

Target The `osquery::readFile` function in osquery/filesystem/filesystem.cpp:70.

Impact An unprivileged system user can coerce osquery into reading arbitrary files, provided they can exploit this race condition while scheduled queries are running.

Description The `readFile` utility function is ubiquitously used throughout osquery, and is intended to provide protections against various types of undesirable read scenarios, such as reading from FIFO files and symbolic links. However, this check contains a time-of-check, time-of-use (TOCTOU) race condition in between the checks of the file and the call to the `ifstream` constructor that follows. The target file could be manipulated in between the calls, allowing a bypass of these checks. As a result, a successful race condition exploit can allow reading of arbitrary files, as the `osqueryd` service typically runs with elevated privileges.

Note that while TOCTOU vulnerabilities are typically easily exploitable by attackers who can control program flow at will (e.g. triggering file reads via commands), this condition is much more difficult to exploit as queries run on a time schedule that an attacker cannot necessarily control, and is thus limited to a time window that occurs when scheduled queries run. In deployments where queries run daily, an attacker will be limited to the times when these queries run, and must wait until the next scheduled run for a failed attempt.

```
72  if (lstat(path.string().c_str(), &file) == 0 && S_ISLNK(file.st_mode)) {
73    if (file.st_uid != 0 && !FLAGS_read_user_links) {
74      return Status(1, "User link reads disabled");
75    }
76  }
77
78  /* snipped for brevity */
79  std::ifstream is(path.string(), std::ifstream::binary | std::ios::ate);
```

Listing 1: Vulnerable code showing the race condition. Note the separate calls to `lstat`, `stat`, and `ifstream` constructor.

Note: This issue was fixed in pull request #1598.

Reproduction Steps A simple method to simulate a successful race condition is to introduce an artificial delay between these checks that would allow replacement of the file before the call to the `ifstream` constructor takes place. For more information on exploiting TOCTOU vulnerabilities, the whitepaper on TOCTOU attacks by iSEC Partners provides numerous examples of how TOCTOU vulnerabilities can be exploited.[3]

Short-Term Mitigation Restructure these checks to operate on the file path and file descriptor rather than only the file path, ensuring that they match. Unfortunately, there is no standard way to do this using the C++ `std::fstream` classes, therefore the checks should be rewritten to use the POSIX APIs (e.g. open, `fstat`).

Recommendation Consider changing the `readFile` implementation to drop privileges to those of the owner. This will prevent access to unauthorized files, despite symbolic links as reads will fail on files that the owner does not have access to.

[3] https://isecpartners.github.io/news/research/2015/03/03/recognizing_preventing_toctou.html

Vulnerability	**Safari Extensions Table Allows Uncontrolled Resource Consumption**
Risk	**Medium** Impact: Low, Exploitability: High
Identifier	NCC-FBOSQ-008
Category	Denial of Service
Target	The `safari_extensions` table implemented in `osquery::tables::genSafariExtensions` at osquery/tables/applications/darwin/browser_plugins.cpp:99.
Impact	A malicious user can place decompression bombs in the Safari extensions directory to purposefully fail queries, potentially preventing other queries from executing. By "blinding" part of scheduled queries, an attacker may be able to prevent important information from being logged.
Description	As implemented, the `safari_extensions` table will list the Safari extensions installed for a given user by scanning the directory where they are stored. These extensions are generally packaged as eXtensible ARchiver (XAR) archives,[4] which are decompressed to extract the `Info.plist` property list file. Because no limits are established, arbitrarily large files can be decompressed, unnecessarily consuming resources. osquery uses a watchdog to prevent queries from causing performance degradation on its host. Once a query exceeds defined limits, the worker performing the queries is killed and restarted. Since queries sharing the same interval or schedule are splayed to execute at slightly different times, queries that did not execute at the time the worker is restarted may not resume. As a result, intentionally failing queries can prevent pending queries from executing. **Note:** This behavior was fixed through other bug fixes, in turn mitigating this issue.
Recommendation	Enforce read limits as the file is decompressed by tracking the uncompressed size while decompressing the target archive entry. If the uncompressed size exceeds the read limit, abort the decompression.

[4] https://developer.apple.com/library/mac/documentation/Darwin/Reference/ManPages/man1/xar.1.html

Vulnerability	**Read Limits Not Enforced for Property Lists**
Risk	**Low** Impact: Low, Exploitability: High
Identifier	NCC-FBOSQ-007
Category	Data Validation
Target	The `osquery::parsePlist` function in osquery/filesystem/darwin/plist.mm:162.
Impact	Arbitrarily large, user-controlled plist files can potentially cause unintended effects. For instance, a large enough plist file can cause queries to fail as described in finding NCC-FBOSQ-008 on the preceding page.
Description	The property list (plist) parsing utility functions are not subject to the same restrictions as general file reads. This is a deviation from the general security model of osquery, where read sizes, particularly those for user-controlled files, are limited according to the parameters provided at runtime. Additionally, since plist files should not be very large by convention,[5] enforcing limits is appropriate. If uncontrolled, unprivileged users can place malicious plist files that can potentially blind a query schedule, as described earlier, when various OS X tables are queried.
Recommendation	Apply the same protections enforced by `osquery::readFile` to subject these files to the same limits.

[5] https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/PropertyLists/AboutPropertyLists/AboutPropertyLists.html

Vulnerability	**Uncaught Exception in JSON Parsing Causes Abort**
Risk	**Low** Impact: Low, Exploitability: Low
Identifier	NCC-FBOSQ-005
Category	Data Validation
Target	The `Pack::initialize` function in osquery/config/packs.cpp:140. See issue #1596.
Impact	Uncaught exceptions cause early program termination, which may be undesirable for longer-running processes and can result in denial of service scenarios.
Description	The `Pack` class parses a JSON configuration containing a list of queries that are scheduled by osquery. While parsing the JSON structure, the `initialize` function expects the "query" key. However, if it is not present, a `Boost` exception is thrown and not handled. This results in early program termination. Currently, this class is used for parsing the `osqueryd` configuration. However, it may result in denial of service scenarios if the class is reused for parsing distributed queries via the TLS plugin interface. This scenario posits a suitably positioned attacker has access to this interface requires compromising several other components.
Recommendation	Move the `initialize` call inside the `try/catch` statement. Alternatively, checking if the key exists may be preferable to avoid an exception context for running instances receiving distributed queries via the TLS plugin interface.

Vulnerability	**Comparison of Integers of Different Signs**
Risk	**Informational** Impact: None, Exploitability: Undetermined
Identifier	NCC-FBOSQ-003
Category	Data Validation
Target	Multiple instances throughout the osquery code base. A full listing can be obtained by performing a clean rebuild without disabled warnings.
Impact	An attacker may be able to cause integer overflows that result in infinite loops or array out-of-bounds memory accesses that may be exploitable in certain cases. This can result in system performance degradation or privilege escalation.
Description	The top-level `CMakeLists.txt` disables warnings for implicit sign conversion. As a result, many instances where integers of different signs were implicitly converted during comparison were unreported. The NCC Group team evaluated and repaired numerous occurrences of this problem throughout the osquery codebase,[6] though was unable to resolve all instances of this issue in the time provided. Some of these violations are possible vulnerabilities with mitigating factors such as memory and time limits, while others are more likely to be simple defects, and a large portion reside in code used for testing. For additional information, and examples, see Appendix B on page 20.
Reproduction Steps	Compile the code without the `-Wno-sign-compare` flag, using the latest available version of the compiler.
Recommendation	Enforce compiler warnings and repair code that causes this warning. As a general rule, this compiler warning should never be disabled and all expression values should use matching types.

[6] See pull request #1585.

Vulnerability **Unsigned Integer Wrap-around**

Risk **Informational** Impact: None, Exploitability: Undetermined

Identifier NCC-FBOSQ-002

Category Data Validation

Target osquery/devtools/printer.cpp and many others throughout the osquery code. See examples below and Appendix C on page 25 for more information.

Impact While unsigned integer wrap-around is well-defined behavior, it is closely associated with security vulnerabilities. Sizes are normally (and correctly) represented as `size_t`, an unsigned type. Unsigned integer wrap-around frequently occurs when calculating sizes. If wrap-around occurs, a too small memory allocation can easily result in an exploitable buffer overflow.

Description NCC Group discovered several instances where unsigned integer wrap-around could occur throughout the osquery code base. These defects were discovered at runtime using the AddressSanitizer feature of Clang 3.7.

Unsigned integer wrap-around occurs in the following code, for example, when `utf8StringSize(col)` is greater than `lengths.at(col)`, and occurs in osquery/devtools/printer.cpp:65:

```
if (lengths.count(col) > 0) {
  int buffer_size = lengths.at(col) - utf8StringSize(col) + 1;
  if (buffer_size > 0) {
    out += std::string(buffer_size, ' ');
  } else {
    out += ' ';
  }
}
```

While the wrap-around behavior appears to be anticipated in this code, the code could just as easily been rewritten to avoid this behavior:

```
if (lengths.count(col) > 0) {
  if (lengths.at(col) + 1 > utf8StringSize(col)) {
    out += std::string(lengths.at(col) - utf8StringSize(col) + 1, ' ');
  } else {
    out += ' ';
  }
}
```

Reproduction Steps Unsigned integer wrap-around defects were detected through dynamic analysis. NCC Group instrumented osquery using the AddressSanitizer feature of Clang 3.7 and ran `make test` to exercise the code. These tests were enabled through the use of the `-fsanitize=unsigned-integer-overflow` flag.

Recommendation NCC Group recommends reviewing instances of possible integer wrap-around, and if possible, eliminating the behaviors. Further instrumented testing, including fuzz testing, should be performed with large integer sizes (e.g., `SIZE_MAX`) to ensure that critical size calculations do not wrap.

Vulnerability	**Undefined Behaviors**
Risk	**Informational** Impact: None, Exploitability: Undetermined
Identifier	NCC-FBOSQ-004
Category	Data Validation
Target	The dynamic pointer cast at osquery/config/config.cpp:272 and others. Instructions on listing additional instances can be found in Appendix E on page 27.
Impact	An undefined behavior can cause errors, crashes, and potentially introduce security vulnerabilities[7].[8]
Description	Undefined behaviors are a major source of vulnerabilities in C and C++. Strictly speaking, undefined behaviors are portability issues. Conforming implementations can deal with undefined behavior in a variety of fashions, such as ignoring the situation completely, with unpredictable results; translating or executing the program in a documented manner characteristic of the environment (with or without the issuance of a diagnostic message); or terminating a translation or execution (with the issuance of a diagnostic message).

The following undefined behavior was reported at runtime using the AddressSanitizer feature of Clang 3.7:

```
include/osquery/registry.h:530:14: runtime error: member call on address 0
    x61300000d940 which does not point to an object of type 'osquery::
    RegistryHelper<osquery::Plugin>'
0x61300000d940: note: object is of type 'osquery::RegistryHelper<osquery::
    ConfigParserPlugin>'
```

This particular problem is likely caused by a design error in the class hierarchy design and enabled by dynamic pointer casts such as the following:

```
parser = std::dynamic_pointer_cast<ConfigParserPlugin>(plugin.second);
```

This undefined behavior occurs frequently at runtime resulting in a large number of runtime errors. It is possible that the code repair might be localized to a few locations in the code base.

For additional information, including instructions to find additional instances, see Appendix E on page 27.

Recommendation	NCC Group recommends regularly testing the code using the `-fsanitize=undefined` flag and exercising the code with existing tests and possibly additional tests, including fuzz testing. All undefined behaviors should be found and eliminated from the code.

[7] http://blog.llvm.org/2011/05/what-every-c-programmer-should-know_14.html
[8] https://www.securecoding.cert.org/confluence/display/c/MSC15-C.+Do+not+depend+on+undefined+behavior

Vulnerability **Memory Leak in Core Application**

Risk **Informational** Impact: None, Exploitability: Undetermined

Identifier NCC-FBOSQ-001

Category Denial of Service

Target The `osquery::tables::xCreate` function in osquery/sql/virtual_table.cpp. For instructions on listing more instances, see Appendix D on page 26.

Impact In addition to being generally undesirable, memory leaks can increase resource usage of osquery subsystems.

Description Memory leaks can be problematic in long running processes such as `osqueryd` because they interfere with normal operation of the product. Four memory leaks were discovered in osquery/sql/virtual_table.cpp. The following is an example:

```
int xCreate(sqlite3 *db,
            void *pAux,
            int argc,
            const char *const *argv,
            sqlite3_vtab **ppVtab,
            char **pzErr) {
  auto *pVtab = new VirtualTable;

  if (!pVtab[9] || argc == 0 || argv [0]
    ==
    nullptr) {
    return SQLITE_NOMEM;

  }
  pVtab->content = new VirtualTableContent;

  PluginResponse response;
  pVtab->content->name = std::string(argv[0]);

  // Get the table column information.
  auto status = Registry::call(
      "table", pVtab->content->name, {{"action", "columns"}}, response);
  if (!status.ok() || response.size() == 0) {
    return SQLITE_ERROR;
  }
```

The naked pointers are not owned by anything that will release the memory on the error cases. The memory referenced by `pVtab` and `pVtab->content` objects will both leak. Because `pVtab` is an automatic or stack variable, in the cases where any of these error conditions are triggered, the lifetime of the `pVtab` object ends when the function returns. Consequently, there is no opportunity to deallocate this memory.

Note that while osquery explicitly protects against resource exhaustion via enforced limits, memory leaks can result in decreased reliability.

Recommendation Consider using `std::unique_ptr`, or one of the other ownership-semantics pointers such as `shared_ptr`.

Appendix A: Vulnerability Field Definitions

The following sections describes the risk rating and category assigned to issues NCC Group identified.

Risk Scale

NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. The risk rating is NCC Group's recommended prioritization for addressing vulnerabilities. Every organization has a different risk sensitivity, so to some extent these recommendations are more relative than absolute guidelines.

Overall Risk

Overall risk reflects NCC Group's estimation of the risk that a vulnerability poses to the target system or systems. It takes into account the impact of the vulnerability, the difficulty of exploitation, and any other relevant factors.

Critical Implies an immediate, easily accessible threat of total compromise.

High Implies an immediate threat of system compromise, or an easily accessible threat of large-scale breach.

Medium A difficult to exploit threat of large-scale breach, or easy compromise of a small portion of the application.

Low Implies a relatively minor threat to the application.

Informational No immediate threat to the application. May provide suggestions for application improvement, functional issues with the application, or conditions that could later lead to an exploitable vulnerability.

Impact

Impact reflects the effects that successful exploitation upon the target system or systems. It takes into account potential losses of confidentiality, integrity and availability, as well as potential reputational losses.

High Attackers can read or modify all data in a system, execute arbitrary code on the system, or escalate their privileges to superuser level.

Medium Attackers can read or modify some unauthorized data on a system, deny access to that system, or gain significant internal technical information.

Low Attackers can gain small amounts of unauthorized information or slightly degrade system performance. May have a negative public perception of security.

Exploitability

Exploitability reflects the ease with which attackers may exploit a vulnerability. It takes into account the level of access required, availability of exploitation information, requirements relating to social engineering, race conditions, brute forcing, etc, and other impediments to exploitation.

High Attackers can unilaterally exploit the vulnerability without special permissions or significant roadblocks.

Medium Attackers would need to leverage a third party, gain non-public information, exploit a race condition, already have privileged access, or otherwise overcome moderate hurdles in order to exploit the vulnerability.

Low Exploitation requires implausible social engineering, a difficult race condition, guessing difficult to guess data, or is otherwise unlikely.

Category

NCC Group groups vulnerabilities based on the security area to which those vulnerabilities belong. This can help organizations identify gaps in secure development, deployment, patching, etc.

Access Controls	Related to authorization of users, and assessment of rights.
Auditing and Logging	Related to auditing of actions, or logging of problems.
Authentication	Related to the identification of users.
Configuration	Related to security configurations of servers, devices, or software.
Cryptography	Related to mathematical protections for data.
Data Exposure	Related to unintended exposure of sensitive information.
Data Validation	Related to improper reliance on the structure or values of data.
Denial of Service	Related to causing system failure.
Error Reporting	Related to the reporting of error conditions in a secure fashion.
Patching	Related to keeping software up to date.
Session Management	Related to the identification of authenticated users.
Timing	Related to race conditions, locking, or order of operations.

Appendix B: Comparison of Integers of Different Signs

As described in finding NCC-FBOSQ-003 on page 14, a major source of defects and potential vulnerabilities in C and C++ code is the implicit conversion of integer types (also called coercion). Warnings for these are disabled by the `-Wno-sign-compare` compile option in `CMakeLists.txt`.

A common problem is that objects used to represent sizes and counts that can never be negative are inappropriately represented as a signed type (typically `int`) when these values should generally be represented as `size_t`.

Additionally, iterators that are used to iterate across these values (say from 0 to size) also need to be of the same type (generally `size_t`) so they have the same range as the bounds.

The following is an example of this problem from osquery/tables/networking/etc_hosts.cpp:37:

```
    for (int i = 1; i < line.size(); ++i) {
```

Listing 2: Line causing implicit sign conversion warning; note that `line.size()` is of type `size_t`.

This problem was fixed by the following change:

```
    for (size_t i = 1; i < line.size(); ++i) {
```

These loops can be exploited by an attacker who can control the size or bounds of the loop, which is a value of type `size_t`. The `size_t` type is an unsigned integer type which represents the size of the largest object which can be allocated on a system and is generally a 32-bit value for a 32-bit address space platform, and a 64-bit value on a 64-bit address space platform. The `int` type is a signed type that is only required by the C Standard to be 16 bits. Even in the case when the `int` type is the same width as `size_t`, because it is a signed value, it has only half the range of its unsigned counterpart. If the attacker provides a value for size that is greater than `INT_MAX` on a given implementation, the index value will wrap-around and assume a negative value. If this value is then used as an array index an out-of-bounds memory access will occur.

The `-Wsign-compare` flag should not be removed and the warning messages addressed. The appropriate repair will vary depending on the exact situation. The best solution is to make sure that all values used in an expression are of the same type. The use of unsigned types is preferred for objects that never take on negative values, such as using `size_t` to represent object sizes.

In some cases, it may be difficult to change the types of formal parameters as it may break the interface. In these cases, the value can be cast to the appropriate type using a C++ style `static_cast`. However, care must be taken when converting from signed to unsigned integer types because negative signed values will be converted into a very large positive value, and when converting from unsigned to signed integer types because sufficiently large values will be represented as negative values. It is important that the bounds are adequately checked in all cases.

For example, in the following situation seen in osquery/sql/virtual_table.cpp:215:

```
  for (size_t i = 0; i < argc; ++i) {
```

Listing 3: Note that `argc` is a signed integer

It is acceptable to change the iterator to a `signed int` type provided that `argc` is properly bounds checked to ensure that it is a positive value and within a defined range. In general, this problem can always be solved with appropriate range checks.

Warnings may be introduced by third-party libraries:

```
osquery/events/linux/audit.cpp: In member function ‘virtual osquery::Status osquery::
    AuditEventPublisher::run()’:
osquery/events/linux/audit.cpp:283:29: warning: comparison between signed and unsigned integer
    expressions [-Wsign-compare]
   if (status_.pid != getpid()) {
```

The definition for `struct audit_status` is in `linux/audit.h` from RedHat.

The `pid` member of `struct audit_status` is declared as `__u32`. The `getpid()` function is declared in `unistd.h` and returns `__pid_t`. IEEE Std 1003.1, 2013 Edition requires that: `blksize_t`, `pid_t`, and `ssize_t` shall be signed integer types. In this regard, the `linux/audit.h` header is incorrect and should be fixed.

This was repaired as follows:

```
diff --git a/osquery/events/linux/audit.cpp b/osquery/events/linux/audit.cpp
index 7ccfadc..571e224 100644
--- a/osquery/events/linux/audit.cpp
+++ b/osquery/events/linux/audit.cpp
@@ -280,7 +280,7 @@ Status AuditEventPublisher::run() {
     }
   }

-  if (status_.pid != getpid()) {
+  if (static_cast<pid_t>(status_.pid) != getpid()) {
     if (control_ && status_.pid != 0) {
       VLOG(1) << "Audit control lost to pid: " << status_.pid;
       // This process has lost control of audit.
```

As previously mentioned, changes to formal parameters can cause problems and it is often easier to work around them. The following problem:

```
osquery/sql/virtual_table.cpp:119:11: warning: comparison of integers of
      different signs: 'int' and 'size_type' (aka 'unsigned long') [-Wsign-compare]
  if (col >= pVtab->content->columns.size()) {
```

Would ideally be repaired as follows:

```
-int xColumn(sqlite3_vtab_cursor *cur, sqlite3_context *ctx, int col) {
+int xColumn(sqlite3_vtab_cursor *cur, sqlite3_context *ctx, size_t col) {
   const BaseCursor *pCur = (BaseCursor *)cur;
```

However, as `col` is a formal parameter, this caused other problems. Because this value, which could be negative, was then used twice as an array index, the following the change was made instead:

```
diff --git a/osquery/sql/virtual_table.cpp b/osquery/sql/virtual_table.cpp
index a60aad7..fcefb4b 100644
--- a/osquery/sql/virtual_table.cpp
+++ b/osquery/sql/virtual_table.cpp
@@ -115,14 +115,15 @@ int xCreate(sqlite3 *db,
int xColumn(sqlite3_vtab_cursor *cur, sqlite3_context *ctx, int col) {
   const BaseCursor *pCur = (BaseCursor *)cur;
   const auto *pVtab = (VirtualTable *)cur->pVtab;
+  size_t ucol = (size_t)col;

-  if (col >= pVtab->content->columns.size()) {
+  if (ucol >= pVtab->content->columns.size()) {
     // Requested column index greater than column set size.
     return SQLITE_ERROR;
   }

-  const auto &column_name = pVtab->content->columns[col].first;
-  const auto &type = pVtab->content->columns[col].second;
+  const auto &column_name = pVtab->content->columns[ucol].first;
```

```
+  const auto &type = pVtab->content->columns[ucol].second;
```

The following changes replaced a `size_t` interator with an `int` interator:

```
@@ -173,7 +174,7 @@ static int xBestIndex(sqlite3_vtab *tab, sqlite3_index_info *pIdxInfo) {

   int expr_index = 0;
   int cost = 0;
-  for (size_t i = 0; i < pIdxInfo->nConstraint; ++i) {
+  for (int i = 0; i < pIdxInfo->nConstraint; ++i) {
     if (!pIdxInfo->aConstraint[i].usable) {
       // A higher cost less priority, prefer more usable query constraints.
       cost += 10;
@@ -212,7 +213,7 @@ static int xFilter(sqlite3_vtab_cursor *pVtabCursor,
   }

   // Iterate over every argument to xFilter, filling in constraint values.
-  for (size_t i = 0; i < argc; ++i) {
+  for (int i = 0; i < argc; ++i) {
     auto expr = (const char *)sqlite3_value_text(argv[i]);
     if (expr == nullptr) {
       // SQLite did not expose the expression value.
```

This solution should be fine, provided these values are with the appropriate range. Alternatively, the signed integer values could have been statically cast to `size_t`.

There are many problems of the form:

```
In file included from osquery/config/tests/packs_tests.cpp:10:
osquery/third-party/gtest-1.7.0/include/gtest/gtest.h:1448:16: warning: comparison
      of integers of different signs: 'const unsigned long' and 'const int' [-Wsign-compare]
  if (expected == actual) {
      ~~~~~~~~ ^  ~~~~~~
osquery/third-party/gtest-1.7.0/include/gtest/gtest.h:1484:12: note: in
      instantiation of function template specialization 'testing::internal::CmpHelperEQ<unsigned
          long,
      int>' requested here
    return CmpHelperEQ(expected_expression, actual_expression, expected,
           ^
osquery/config/tests/packs_tests.cpp:65:3: note: in instantiation of
      function template specialization 'testing::internal::EqHelper<false>::Compare<unsigned long,
      int>' requested here
  EXPECT_EQ(tree.count("packs"), 1);
  ^
osquery/third-party/gtest-1.7.0/include/gtest/gtest.h:1979:53: note: expanded from
      macro 'EXPECT_EQ'
                    EqHelper<GTEST_IS_NULL_LITERAL_(expected)>::Compare, \
                                                               ^
osquery/third-party/gtest-1.7.0/include/gtest/gtest_pred_impl.h:162:23: note:
      expanded from macro 'EXPECT_PRED_FORMAT2'
  GTEST_PRED_FORMAT2_(pred_format, v1, v2, GTEST_NONFATAL_FAILURE_)
                      ^
osquery/third-party/gtest-1.7.0/include/gtest/gtest_pred_impl.h:147:17: note:
      expanded from macro 'GTEST_PRED_FORMAT2_'
  GTEST_ASSERT_(pred_format(#v1, #v2, v1, v2), \
                ^
osquery/third-party/gtest-1.7.0/include/gtest/gtest_pred_impl.h:77:52: note:
      expanded from macro 'GTEST_ASSERT_'
  if (const ::testing::AssertionResult gtest_ar = (expression)) \
                                                   ^
1 warning generated.
```

The largest problem in this instance is that `EXPECT_EQ` is compared to objects such as `count` and `size` which are unsigned with integer literals such as 0, 1, and 2 which if undecorated are by default of type `signed int`. The simple solution in most cases is to use the "U" unsigned suffix to make them unsigned, e.g.: `0U`, `1U`, and `2U`. Such changes appear as follows:

```
diff --git a/osquery/config/tests/packs_tests.cpp b/osquery/config/tests/packs_tests.cpp
index ab2c1c8..5a1971f 100644
--- a/osquery/config/tests/packs_tests.cpp
+++ b/osquery/config/tests/packs_tests.cpp
@@ -62,7 +62,7 @@ pt::ptree getPackWithFakeVersion() {

TEST_F(PacksTests, test_parse) {
   auto tree = getExamplePacksConfig();
-  EXPECT_EQ(tree.count("packs"), 1);
+  EXPECT_EQ(tree.count("packs"), 1U);
}

TEST_F(PacksTests, test_should_pack_execute) {
@@ -119,7 +119,7 @@ TEST_F(PacksTests, test_check_version) {

TEST_F(PacksTests, test_schedule) {
   auto fpack = Pack("foobar", getPackWithDiscovery());
-  EXPECT_EQ(fpack.getSchedule().size(), 1);
+  EXPECT_EQ(fpack.getSchedule().size(), 1U);
}

TEST_F(PacksTests, test_discovery_cache) {
@@ -131,16 +131,16 @@ TEST_F(PacksTests, test_discovery_cache) {

   auto c = Config();
   c.addPack(pack);
-  int query_count = 0;
-  for (int i = 0; i < 5; i++) {
+  size_t query_count = 0;
+  for (size_t i = 0; i < 5; i++) {
     c.scheduledQueries(
         ([&query_count](const std::string& name, const ScheduledQuery& query) {
           query_count++;
         }));
   }
-  EXPECT_EQ(query_count, 5);
+  EXPECT_EQ(query_count, 5U);

-  int pack_count = 0;
+  size_t pack_count = 0U;
   c.packs(([&pack_count](Pack& p) {
     pack_count++;
     EXPECT_EQ(p.getStats().total, 5);
@@ -148,7 +148,7 @@ TEST_F(PacksTests, test_discovery_cache) {
     EXPECT_EQ(p.getStats().misses, 1);
   }));

-  EXPECT_EQ(pack_count, 1);
+  EXPECT_EQ(pack_count, 1U);
}

TEST_F(PacksTests, test_discovery_zero_state) {
```

Multiple changes were made in multiple source code files to eliminate these errors.

B.1 Unrepaired Errors

While NCC Group repaired several instances of of different signedness integer comparison as mentioned above, the following errors have not yet been corrected:

- osquery/sql/virtual_table.cpp:119
- osquery/sql/virtual_table.cpp:176
- osquery/sql/virtual_table.cpp:215
- osquery/tables/networking/linux/routes.cpp:62
- osquery/core/watcher.cpp:278
- osquery/core/watcher.cpp:301
- osquery/events/events.cpp:136
- osquery/filesystem/filesystem.cpp:102
- osquery/events/linux/tests/inotify_tests.cpp:76
- osquery/sql/virtual_table.cpp:119
- osquery/sql/virtual_table.cpp:176
- osquery/sql/virtual_table.cpp:215
- osquery/tables/networking/linux/routes.cpp:62
- osquery/core/watcher.cpp:278
- osquery/core/watcher.cpp:301
- osquery/events/events.cpp:136
- osquery/filesystem/filesystem.cpp:102
- osquery/events/linux/tests/inotify_tests.cpp:76

Appendix C: Unsigned Integer Wrap-around

Unsigned integer wrap-around is actually well-defined behavior in C and C++. The C Standard, 6.2.5, paragraph 9,[10] states:

> A computation involving unsigned operands can never overflow, because a result that cannot be represented by the resulting unsigned integer type is reduced modulo the number that is one greater than the largest value that can be represented by the resulting type.

This behavior is more informally called unsigned integer wrapping or wrap-around. Unsigned integer operations can wrap if the resulting value cannot be represented by the underlying representation of the integer.

Integer values must not be allowed to wrap, especially if they are used in any of the following ways[11]:

- Integer operands of any pointer arithmetic, including array indexing
- The assignment expression for the declaration of a variable length array
- The postfix expression preceding square brackets `[]` or the expression in square brackets `[]` of a subscripted designation of an element of an array object
- Function arguments of type `size_t` or `rsize_t` (for example, an argument to a memory allocation function)
- In security-critical code

The examples of unsigned integer wrap-around reported in this section as well as finding NCC-FBOSQ-002 on page 15 were discovered at runtime using the AddressSanitizer feature of Clang 3.7.

```
/usr/include/boost/lexical_cast.hpp:1212:70: runtime error: unsigned integer overflow: 0 - 1 cannot
    be represented in type 'unsigned long'
SUMMARY: AddressSanitizer: undefined-behavior /usr/include/boost/lexical_cast.hpp:1212:70 in
```

```
/usr/include/boost/asio/detail/timer_queue.hpp:196:33: runtime error: unsigned integer overflow: 0 -
     1 cannot be represented in type 'unsigned long'
SUMMARY: AddressSanitizer: undefined-behavior /usr/include/boost/asio/detail/timer_queue.hpp:196:33
    in
```

```
/usr/include/boost/xpressive/detail/utility/sequence_stack.hpp:61:16: runtime error: unsigned
    integer overflow: 0 - 1 cannot be represented in type 'std::size_t' (aka 'unsigned long')
SUMMARY: AddressSanitizer: undefined-behavior /usr/include/boost/xpressive/detail/utility/sequence_
    stack.hpp:61:16 in
```

```
osquery/third-party/sqlite3/sqlite3.c:56558:20: runtime error: unsigned integer overflow: 0 - 1
    cannot be represented in type 'u32' (aka 'unsigned int')
SUMMARY: AddressSanitizer: undefined-behavior osquery/third-party/sqlite3/sqlite3.c:56558:20 in
```

```
osquery/devtools/printer.cpp:65:41: runtime error: unsigned integer overflow: 2 - 6 cannot be
    represented in type 'unsigned long'
SUMMARY: AddressSanitizer: undefined-behavior osquery/devtools/printer.cpp:65:41 in
```

[10] ISO/IEC. Programming Languages—C, 3rd ed (ISO/IEC 9899:2011). Geneva, Switzerland: ISO, 2011.
[11] Seacord, Robert C. CERT® C Coding Standard, Second Edition, The: 98 Rules for Developing Safe, Reliable, and Secure Systems, 2nd Ed. Boston: Addison-Wesley, 2014

Appendix D: Memory Leaks

To detect findings such as the one in finding NCC-FBOSQ-001 on page 17, NCC Group installed the Clang 3.7 compiler and performed various tests using the following settings:

- `add_compile_options(-fsanitize=leak -fsanitize=address)`
- `add_compile_options(-fsanitize=undefined -fsanitize=unsigned-integer-overflow)`
- `add_compile_options(-fsanitize=integer)`
- `add_compile_options(-fsanitize-memory-track-origins=2 -fsanitize=memory)`

In all cases, the following compile options were also used to produce the output:

```
add_compile_options(-g -O1 -fno-omit-frame-pointer -fno-optimize-sibling-calls)
```

and the following link was added to `tools/provision/ubuntu.sh`:

```
sudo ln -sf /usr/bin/llvm-symbolizer-3.7 /usr/bin/llvm-symbolizer
```

to ensure that `llvm-symbolizer` was in the execution path to produce diagnostics with human readable symbols.

Of these four sets of tests, all were successful except the memory test, which the NCC Group team was unable to build because of unresolved external references. It is recommended that the Facebook team attempt to complete this test.

Appendix E: Undefined Behaviors

The C Standard, subclause 3.4.3, defines undefined behavior as "behavior, upon use of a nonportable or erroneous program construct or of erroneous data, for which the standard imposes no requirements".[12]

Annex J, subclause J.2, "Undefined behavior" of the C Standard enumerates the circumstances under which the behavior of a program is undefined.

Behavior can be classified as undefined for the following reasons:

- To give the implementor license not to catch certain program errors that are difficult to diagnose.
- To avoid defining obscure corner cases that would favor one implementation strategy over another.
- To identify areas of possible conforming language extension: the implementor may augment the language by providing a definition of the officially undefined behavior.

Undefined behavior in C++ is defined in the same manner as C, but the C++ language adds additional undefined behaviors.[13]

Undefined behaviors such as the one described above were detected through dynamic analysis. NCC Group instrumented osquery using the AddressSanitizer feature of Clang 3.7 and ran `make test` to exercise the code. These tests were enabled through the use of the `-fsanitize=undefined` flag.

Exercising the tests yields output similar to the following:

```
osquery/include/osquery/registry.h:530:14: runtime error: member call on address 0x61300000d940
    which does not point to an object of type 'osquery::RegistryHelper<osquery::Plugin>'
0x61300000d940: note: object is of type 'osquery::RegistryHelper<osquery::ConfigParserPlugin>'
 01 00 80 53  48 48 4c 02 00 00 00 00  a8 df 00 00 40 60 00 00  01 be be be be be be be  be be be be
              ^~~~~~~~~~~~~~~~~~~~~~~
              vptr for 'osquery::RegistryHelper<osquery::ConfigParserPlugin>'
SUMMARY: AddressSanitizer: undefined-behavior osquery/include/osquery/registry.h:530:14 in
osquery/include/osquery/registry.h:530:14: runtime error: member call on address 0x61300000d940
    which does not point to an object of type 'osquery::RegistryHelper<osquery::Plugin>'
0x61300000d940: note: object is of type 'osquery::RegistryHelper<osquery::ConfigParserPlugin>'
 01 00 80 53  48 48 4c 02 00 00 00 00  a8 df 00 00 40 60 00 00  01 be be be be be be be  be be be be
              ^~~~~~~~~~~~~~~~~~~~~~~
              vptr for 'osquery::RegistryHelper<osquery::ConfigParserPlugin>'
```

[12] ISO/IEC. Programming Languages—C, 3rd ed (ISO/IEC 9899:2011). Geneva, Switzerland: ISO, 2011.
[13] Seacord, Robert C. CERT® C Coding Standard, Second Edition, The: 98 Rules for Developing Safe, Reliable, and Secure Systems, 2nd Ed. Boston: Addison-Wesley, 2014

Under certain circumstances, compilers are permitted to omit the copy and move constructors of class objects even if copy/move constructor and the destructor have observable side-effects. Clang 3.7 warns on calls to `std::move` (typically in return statements) that prevent copy elision[14] from being performed:

```
72      return std::move(tables::genLastAccess(request));
```

Listing 4: Code in `additional_amalgamation.cpp:40` that results in a pessimizing move.

Code should be compiled with Clang 3.7 or later that warns on pessimizing moves, with the `-Wpessimizing-move` flag. Offending calls should be removed.

Many of these calls reside in the amalgamation code generated at `build/<platform>/generated/additional_amalgamation.cpp` in addition to osquery/remote/utility.h:59 as used by the TLS plugins at osquery/config/plugins/tls.cpp.

[14] https://en.wikipedia.org/wiki/Copy_elision