Lantern Client Application Security Assessment

Lantern

April 26, 2017 – Version 1.4

Document Change Log

Version	Date	Change
1.0	2016-08-15	Initial draft delivered to Lantern on July 29
1.1	2016-09-12	Sent for internal peer review
1.2	2016-09-22	Ready for Lantern
1.3	2016-09-26	Minor grammar fixes
1.4	2017-04-26	Final minor fixes

Executive Summary

Synopsis

During the summer of 2016, Lantern and Open Technology Fund engaged NCC Group to conduct a security assessment of Lantern's architecture and clients. Lantern provides a proxy intended for use to bypass censorship. This assessment was open ended and time-boxed, providing a best-effort security analysis in a fixed amount of time. Source code review was the primary method of assessment. One consultant focused on code review from a vulnerability discovery perspective, and a second consultant focused on systemic and architectural issues.

Scope

NCC Group's evaluation included:

- Desktop Client: The main component of the software is the cross-platform Lantern client. The client is written principally in Go with some components in other languages, including C, C++, Objective-C, and JavaScript.
- Proxy Architecture: While the server code itself was not in scope for the assessment, NCC Group did assess the overall architecture of the solution at a high level.

This application is intended for use in countries where the Internet is censored and therefore its threat model includes risks related to attribution and privacy attacks beyond just software security vulnerabilities. Included in that threat model are well-resourced attackers with advanced capabilities such as reading or modifying HTTPS traffic unbeknownst to the targets.

Key Findings

The assessment uncovered a variety of issues that could negatively affect Lantern users. Some of the more notable issues were:

- Any visited website can detect a running instance of Lantern, even if Lantern was not actively attempting to proxy the connection. It is worth noting that Lantern specifically excludes user detection from their threat model.
- Several issues around updates, to both binaries and configurations, could allow a well-prepared attacker to subvert the update process. In most cases, the attacker would need network position between the proxy CDN and the web server hosting the update. Binary updates could be manipulated to downgrade a Lantern client to an earlier legitimate version.
- Some issues revolve around the ways the client decides whether to proxy or not. Because these methods are easily discoverable by reviewing source, and because the default is to attempt to connect directly, censors could choose to react differently to Lantern and non-Lantern clients, or simply attempt to stay ahead of Lantern's censorship detection.
- Proxies can be abused in a variety of ways, such as to cost Lantern money in hosting fees, perform click fraud, overload Lantern proxies, and sour relations with CDNs or partners.
- An attempt to obfuscate local configuration files was not effective. The feature, meant to protect users from unknowingly publishing sensitive information about themselves, was a short-term fix but not a solution.

Limitations

NCC Group achieved adequate coverage of the Go code, which forms the backbone of the Lantern client. Some related components were not evaluated:

- Server-side components were not in scope for the assessment.
- The project relies on many third-party libraries, including some written in C and some with significant network attack surface. These libraries were not thoroughly evaluated, though some were briefly examined.
- The Android client is significantly different than the desktop clients and these differences were not fully evaluated.

Strategic Recommendations

Decentralize – Despite some claims to the contrary, Lantern's proxy infrastructure is centralized and does not currently contain peer-to-peer components. Peer-to-peer architecture would reduce single points of failure and hedge against loss of proxy capacity due to political pressure on partners.

Additional censorship evasion techniques – Continue development towards additional anti-censorship techniques and rely less on domain fronting as the primary method of traffic obfuscation.

Remove C dependencies – Some third-party code used by Lantern with network attack surface is written in C, which increases the risk of memory corruption vulnerabilities. In particular, BadVPN is used in the Android client. NCC Group did not thoroughly assess BadVPN or other third-party code but recommends it be removed to reduce attack surface and help guard against memory corruption attacks against the client.

Project Dashboard

Target Metadata

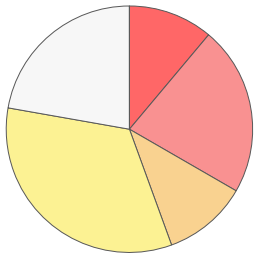
Name	Lantern Client
Type	Native proxy application
Platforms	Go, with some other components (C, Java)
Environment	Local client

Engagement Data

Type	Application security assessment
Method	Code review, dynamically assisted
Dates	2016-07-11 to 2016-07-29
Consultants	2
Level of effort	6 person-weeks

Vulnerability Breakdown

Critical Risk issues	1
High Risk issues	2
Medium Risk issues	1
Low Risk issues	3
Informational issues	2
Total issues	9



Category Breakdown

Access Controls	2
Configuration	1
Cryptography	3
Data Exposure	3

Component Breakdown

Update system	3
Server on localhost:16823	1
Proxy servers	1
Proxy	2
localhost proxy	1
Configuration storage	1

Key

Critical High Medium Low Informational

Test Details

Synopsis

Lantern and Open Technology Fund engaged NCC Group to perform a security review of the Lantern system. Lantern is a client program that attempts to detect sites being actively censored and then route around the censorship by using a variety of proxy methods. The engagement was a six-week effort split between two consultants. One consultant focused on reviewing the Go code and some basic dynamic testing. The second consultant looked for systemic or architectural issues that could compromise the security of Lantern users or the system as a whole. There was some crossover in aspects of the review between the two consultants.

Testing was focused on both the client software itself and the overall architecture of the system, but server-side applications were not in scope for the assessment.

Testing Methods

Unlike most security assessments, this review also considered privacy breaches and other similar areas of concern when software is used to hide activities from powerful entities, such as governments. Of particular note, NCC Group considered:

- Distribution – How can users of the software be assured they are running the software they expected? This also includes features like automated updates.
- Fingerprinting – Can a Lantern user be identified as such?
- Resistance to blocking – An anti-censorship tool that is itself censored is not of great use.
- Attribution Protection – Can a user of the proxy be positively identified as a specific person? Lantern, like other censorship evasion networks (e.g. Psiphon), do not purport to provide any builtin network anonymity measures but should ensure that its users' privacy is protected.

In addition to the above, NCC Group examined the client-side source code for common types of vulnerabilities, mostly via code review. This review included, but was not limited to, identifying:

- Memory Corruption Vulnerabilities – Lantern is written primarily in Go, which itself has builtin defenses to memory corruption vulnerabilities (barring the use of specific unsafe pointer types and the like). Lantern relies on libraries that were built from memory-unsafe languages and includes a small amount of original code in C, C++, and Objective-C. Security review included examination of all use of unsafe pointer types and examination of C, C++, and Objective-C code that was original to Lantern. Libraries built from memory-unsafe languages were not specifically in scope, but NCC Group did perform a cursory examination of a few key libraries.
- SQL Injection Vulnerabilities – Like all languages, Go is vulnerable to SQL injection attacks when a query string is concatenated with user data. NCC Group examined the code for such SQL calls.
- Command Injection Vulnerabilities – Like all languages, Go is vulnerable to command injection when user input is concatenated or otherwise combined with command-line statements or arguments.
- Cryptographic Weaknesses – The use of properly encrypted communications between endpoints is a cornerstone of the application and in scope for this assessment. In Lantern's case, TLS certificate pinning was in use, so examination of the pinning methods was needed. The auto-update feature and its signature system were also carefully examined.
- Proxy Abuse – Can hostile actors abuse the proxy services, either by routing traffic through legitimate users' proxies or other, more direct means?

Threats

It is important to note that some of the attackers of the Lantern system are likely to be nation-states or similarly powerful groups. Thus, it is worth considering that attackers will have capabilities that in other scenarios might seem unrealistic. For example, we assumed that attackers can always put themselves in the network path between the user and Lantern servers. We also assumed the existence of attackers who are in control of root certificate authorities, or have other means to subvert the traditional TLS Certificate Authority/Public Key Infrastructure.

Best Practices Evaluation

This section summarizes the architecture and design qualities of Lantern. The specific threats and heightened security measures employed by a censorship evasion tool are examined below.

Rating Explanation

- Satisfactory: Meets or exceeds industry best practice
- Fair: May not be in total compliance with best practices but no directly actionable issues were identified
- Needs improvement: Fails to comply with best practices, and contains actionable flaws in this area

Code and Memory Safety

As designed:	As implemented:
Satisfactory	**Fair**

Best Practice: Because they have a significant network-accessible attack surface, tools such as Lantern should take great effort to ensure that the source code does not contain memory corruption vulnerabilities or other vulnerabilities such as command injection.

Evaluation: The version of Lantern reviewed is written in Go, which is resistant to memory corruption vulnerabilities. Some C code remains in libraries and specialty functions. NCC Group did not identify any vulnerable C code but did not thoroughly examine third-party libraries in use.

Testing reviewed for a variety of common application vulnerabilities (i.e. command injection) but no examples were identified during the testing period.

Recommendation: Going forward, continue to replace C code and libraries with Go (or other languages with intrinsic security qualities).

Communication Security

As designed:	As implemented:
Fair	**Needs improvement**

Best Practice: Transport Layer Security (TLS) should be used for all connections, and public keys included in the application distribution should be used to verify that a hostile actor with access to a Certificate Authority (CA) cannot intercept and read or modify traffic. Pinned certificates should be pinned as close to the leaf certificate as is practical, and the pinning should terminate on the application server or similar servers controlled directly by the project.

When data is passed to a recipient via the server, the data should be encrypted between recipients, and when it is passed from a client to a server, TLS with Ephemeral Keys provides end-to-end security.

Evaluation: All communications between the client and Lantern network use TLS. In some cases, these communications were pinned to the proxy endpoint, but this leaves a gap between the proxy server/CDN and the final destination. Also, during the initial setup, certificates were not pinned at all.

Recommendation: At the very least, all TLS connections should be pinned to the proxy, as the largest threat to this type of application comes from a censoring organization (such as an ISP) who controls some part of the network close to the client. Ideally, additional pinning could be used to the application server when downloading updates, configurations, etc.

Resistance to Blocking

As designed:	As implemented:
Satisfactory	**Fair**

Best Practice: A system designed to bypass censorship might have to deal with censoring entities who attempt to disrupt the system's ability to deliver uncensored content. Therefore, the anti-censorship system needs to be robust against Denial of Service attacks and have agile or obfuscated points of entry to prevent users from being blocked before they can even get to the proxy.

Evaluation: Lantern's system for preventing blockage via hiding in CDN traffic (domain fronting[1]) is proven to be generally effective with a supportive CDN. Pinned TLS traffic through the CDN also reduces the risk of blocking via deep packet inspection. However, in the past, censorship systems have found ways of blocking these connections without incurring full "collateral damage" of blocking the fronting domain. There have been examples of blocking based on client TLS signatures[2] as well as CDNs being affected by political or monetary pressure.[3] Additionally, CDNs provide an easy single point of breach where a hostile government could view all incoming and outgoing Lantern traffic.

Recommendation: Censorship evasion will continue to be an arms race requiring agile changes to keep up with censorship techniques. Continue efforts to support additional traffic obfuscation systems. Consider integrating additional pluggable transports[4] or making efforts to decentralize.

Build Process and Distribution

As designed:	As implemented:
Fair	Fair

Best Practice: For open source projects of this type, we expect to see build servers that are somewhat hardened, signed binaries with keys under strict control, along with other attempts to ensure the build process is consistent and secure. There should be an automatic update feature that keeps the software current, and that feature should have similar protections.

Additionally, having reproducible builds helps the internet at large evaluate that what was in source code is what was delivered in packaged binaries.

Evaluation: Builds are given a SHA256 hash and then signed using RSA. This signature and hash are validated upon download. Some corner cases allow tampering with the autoupdate process.

No reproducible builds are provided.

Recommendation: Fix the autoupdate issues (see finding NCC-LANT16-002) described later in the document and provide reproducible builds at a later date.

Fingerprinting

As designed:	As implemented:
Satisfactory	Fair

Best Practice: Ideally, censoring parties should not be able to tell a "normal" user from a user bypassing the censorship. This prevents retaliation against users circumventing the censorship and would make blocking the bypass more difficult.

Evaluation: Though the CDN proxy does prevent much traffic analysis, flaws discovered in the system allow any website the user visits to determine if Lantern is running on the client (though not necessarily whether the visit was proxied or not). This could be used to highlight Lantern users and flag or block their connections.

Recommendation: Fix the issues allowing websites to discover a running Lantern proxy. See finding NCC-LANT16-001 for additional information.

[1] http://www.icir.org/vern/papers/meek-PETS-2015.pdf
[2] https://groups.google.com/forum/#!msg/traffic-obf/BpFSCVgi5rs/nCqNwoeRKQAJ
[3] https://groups.google.com/forum/#!topic/traffic-obf/XX-g1yERttE
[4] https://obfuscation.github.io/

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
Communication for configuration updates is unencrypted behind proxies	007	Critical
Visited sites can detect Lantern users	001	High
Proxy servers could be abused	008	High
Autoupdate signature does not contain version identifier	002	Medium
Publicly known methods for detection of blocked sites	003	Low
Opportunistic proxied routing	004	Low
Unrestricted open proxy service	009	Low
Configurations on disk are stored with trivial obfuscation	005	Informational
Lack of certificate pinning in connection to autoupdate server	006	Informational

Vulnerability Details

Vulnerability	**Communication for configuration updates is unencrypted behind proxies**
Risk	**Critical** Impact: High, Exploitability: Low
Identifier	NCC-LANT16-007
Category	Cryptography
Component	Update system
Location	getlantern/flashlight/config (private location)
Impact	An attacker who can intercept traffic between the proxy servers and the configuration server can read and modify traffic, including but not limited to configuring different proxy servers, proxy lists and root certificates to trust for proxy servers.
Description	When updating configuration, the Lantern application makes requests through the configured proxies. However, the URL to fetch the configuration does not make use of TLS. If an attacker is able to intercept traffic between the exit proxy and the configuration server, the attacker will be able to read and modify traffic. By modifying configuration server traffic, an attacker would be able to push new configurations to clients including custom proxy servers and trusted root certificate authorities for these proxies, compromising any protections provided by the application. Although traffic between proxy servers and configuration servers should in all circumstances be routed outside of any censoring network, plaintext communications behind the proxies remain vulnerable to any attackers along the plaintext communication route.
Recommendation	Encrypt the communication between the proxy servers and configuration servers by using TLS and certificate pinning.

Vulnerability	**Visited sites can detect Lantern users**
Risk	**High** Impact: High, Exploitability: High
Identifier	NCC-LANT16-001
Category	Data Exposure
Component	Server on localhost:16823
Location	http://127.0.0.1:16823/proxy_on.pac?1468350415635649072
Impact	Any website visited by a Lantern user while Lantern is running can detect the presence of Lantern. The impact of this on users of Lantern living under repressive regimes is not known, but at the very least one could assume that a list of Lantern users could be compiled to be referred to later by authorities. NCC Group consulted with third-parties to ascertain if there were known cases of imprisonment or persecution of people who use censorship circumvention technologies. To NCC Group's knowledge there are no widely known instances of punishment targeting censorship circumventers, unless those circumventers were also easily identified as having political, anti-government goals. It is the Group's understanding that Lantern is not marketed or commonly used in this way.
Description	Any site can use JavaScript to attempt to request the URL above. Although the browser's Content Security Policy should prevent the site from reading the content of the response, it can detect whether a response was received or not. This was tested on Safari and will likely work in other browsers as well. It does not matter whether Lantern has actively attempted to evade censorship for that site or not as this port is always open.
Reproduction Steps	See Appendix B for example code to detect Lantern from a web server.
Recommendation	All ports opened by Lantern should be determined randomly upon startup, and the name of the proxy file served should also have a significant random component.

Vulnerability	**Proxy servers could be abused**
Risk	**High** Impact: High, Exploitability: Medium
Identifier	NCC-LANT16-008
Category	Access Controls
Component	Proxy servers
Impact	An attacker could attempt to flood Lantern's proxy or update servers with traffic to impede use or incur excessive bandwidth fees for Lantern.
Description	Because Lantern users can be identified, and because sites are directly loaded by default, an organization which commonly receives significant direct (non-proxied) traffic could add Javascript to their pages that make unwanted requests across the proxy. In cases where Lantern pays for this traffic, a significant monetary cost could be incurred. Even in cases where bandwidth has no significant cost to Lantern, this traffic could simply be used to slow down the system, either by overloading servers or by flooding the proxy pipe for each individual user. If the enumeration (see finding NCC-LANT16-001) and open proxy (see finding NCC-LANT16-009) issues are fixed, an attacker could perform these same attacks simply by running many instances of Lantern, or simply pretend to with custom software that talks to Lantern servers. This is less efficient as the attacker would have to host the machines and provide bandwidth, but well-resourced attackers could probably still cause significant damage. This is largely a conceptual attack at this point, NCC Group did not provide a proof-of-concept but believes the attack to be practical.
Recommendation	It is not likely that this behavior can be fixed in a meaningful way on the client side. Implement server-side monitoring that tracks usage patterns and alerts on anomalies. Devise a plan that can blackhole malicious requests using any available indicators, whether at the IP, TLS or HTTP layer.

Vulnerability	**Autoupdate signature does not contain version identifier**
Risk	**Medium** Impact: High, Exploitability: Medium
Identifier	NCC-LANT16-002
Category	Cryptography
Component	Update system
Location	https://github.com/getlantern/go-update/update.go `verifySignature` function
Impact	An attacker who can modify autoupdate communications can downgrade the connecting client by submitting an old update with a correct signature.
Description	The Lantern client uses automatic updating methods to discover and apply updates. It achieves this by making HTTPS requests to the update server containing the client's platform information, and receives back information about the latest available update for its platform. This information includes a URL to download a patch from, the new version, a cryptographic signature for the resulting binary, and other associated information. If the version identifier is newer than the current version the client will connect to the update URL to download the patch, and will verify the signature against a hard-coded public key before applying the patch. If an attacker is either able to pose as a malicious update server or is able to intercept communications, the attacker can reply with an newer version identifier but supply an older binary and its associated signature to the client. This is possible as the binaries and signatures are given to all clients and can be recorded, and the version identifier is not included in the cryptographic signature. Exploitation of this vulnerability requires being able to intercept the TLS connection behind the proxy. This connection does not use certificate pinning, and so is vulnerable to an attacker who can sign the TLS certificate with a valid certificate that chains to a default trusted CA. Although this is a significant requirement, national censored networks often have this capacity.
Reproduction Steps	This finding was discovered through a code review of the autoupdate functionality. Relevant packages are at: • github.com/getlantern/go-update • github.com/getlantern/autoupdate • github.com/getlantern/flashlight/autoupdate In particular the functionality responsible for creating the new patched binary and verifying signatures is located at `github.com/getlantern/go-update/update.go:318 FromStream`. To perform a downgrade, an attacker would have to create a custom patch for a particular client version that would result in an older client version. The attacker could then respond with this patch and the signature associated with the older client version to incoming requests.
Recommendation	The cryptographic signature should apply to the entire update package. This would fix the downgrade problem and also remove the possibility for similar attacks using unsigned portions of the update package.

Vulnerability	**Publicly known methods for detection of blocked sites**
Risk	**Low** Impact: Low, Exploitability: High
Identifier	NCC-LANT16-003
Category	Data Exposure
Component	Proxy
Location	github.com/getlantern/detour
Impact	A censoring network can prevent censored websites from being automatically added to the list of sites to access via proxy.
Description	In the default configuration, not all traffic is proxied by the Lantern application. The configuration files contain a list of known blocked sites for which traffic is proxied. For websites not on this list, ad-hoc detection of censorship is done using country specific detectors. The method of detection can always be determined by running or analyzing the client Lantern application even if source code is not available. As these detection methods are publicly known, a censoring network can avoid detection by changing the result of blocked websites.
Reproduction Steps	This finding was discovered through a code review. The relevant package is github.com/getlantern/detour.
Recommendation	Proxy all connections by default and use a whitelist for websites that are known to normally not need Lantern to connect to it. Alternatively, allow a user to easily add a website to a list of sites to be proxied.

Vulnerability	**Opportunistic proxied routing**
Risk	**Low** Impact: Low, Exploitability: Low
Identifier	NCC-LANT16-004
Category	Configuration
Component	Proxy
Location	github.com/getlantern/detour
Impact	Initial connections to websites that are not in the default "to be proxied" list are served directly, allowing censoring networks to determine if a user attempts to access censored websites.
Description	Initial connections to websites not on the configuration file's list of websites to be proxied, will not initially be proxied. For these connections, an attacker with access to network traffic can see destination IP addresses and, additionally, any plaintext traffic can be viewed and modified. TLS traffic which does not use certificate pinning can be viewed and modified by state censorship networks who can issue valid certificates from publicly accepted root certificate authorities.
Recommendation	Proxy all connections by default and use a whitelist for websites known to not be blocked.

Vulnerability	**Unrestricted open proxy service**
Risk	**Low** Impact: Low, Exploitability: Medium
Identifier	NCC-LANT16-009
Category	Access Controls
Component	localhost proxy
Impact	An attacker able to make local network connections can use Lantern's open proxy to exhaust the allotted bandwidth for the user thus causing a denial-of-service attack.
Description	Especially on Android, but likely on desktop clients as well, third party applications (and probably websites visited in a browser) can push requests through Lantern's local proxy. When the connection through Lantern is metered (at the time of this writing, Lantern provided 800MB of high-speed data per month), a third-party could intentionally pass a large amount of traffic through the proxy simply to exhaust the user's allotment and thereby blocking access to Lantern's network.
Recommendation	Consider adding restrictive measures to the proxy service so that other applications on the system are not able to use the proxy. As this is a cross-platform application, these restrictions may need to be platform specific. For some platforms, the proxy service could support authentication requiring credentials to use the proxy. For Android, these credentials could be stored in the application's protected storage location. For Linux, App-Armor or SELinux rules could be used to define which Unix groups are allowed to interact with the port.

Vulnerability	**Configurations on disk are stored with trivial obfuscation**
Risk	**Informational** Impact: Low, Exploitability: High
Identifier	NCC-LANT16-005
Category	Data Exposure
Component	Configuration storage
Location	github.com/getlantern/rot13 github.com/getlantern/yamlconf
Impact	Persistent configurations are weakly obfuscated and can easily be read or modified.
Description	Configuration is stored in YAML files on disk. This configuration is saved after performing ROT13 on the data. The intention behind using ROT13 when saving the configuration file was explained as a short-term fix to obscure the information. Although this feature was not heavily relied upon as a security control, it should be removed completely.
Reproduction Steps	1. Navigate to the local configuration storage directory. On OSX this is at `~/Library/Application Support/Lantern`. 2. Open the lantern-version.yaml file in a text editor to see the file is not plaintext, and note the repeating sequences. 3. Determine through further analysis that the file is processed with ROT13. Alternatively, view the relevant source at github.com/getlantern/yamlconf and github.com/getlantern/flashlight/config.
Recommendation	If encryption or authentication is intended, use proper cryptographic primitives and a program-launch flow that only recovers a configuration file upon startup when the user enters a pass phrase. In any case, take care to define the threats to defend against (e.g. the threat of malware accessing memory locations) before determining which solution to implement.

Vulnerability	**Lack of certificate pinning in connection to autoupdate server**
Risk	**Informational** Impact: High, Exploitability: Low
Identifier	NCC-LANT16-006
Category	Cryptography
Component	Update system
Location	github.com/getlantern/autoupdate
Impact	An advanced attacker who can intercept network traffic behind the proxy servers, and can issue certificates with a valid certificate chain can intercept and modify traffic to the autoupdate server.
Description	All connections to the autoupdate server connect through the proxy servers defined in the local configuration. This initial connection from local machine to proxy server is encrypted with TLS and uses pinned certificates. The connection from the proxy server and the autoupdate server is also over TLS, but the root certificate authority will only be verified if the current client has the configurable value `CloudConfigCA` set. The default configurations do not set this value. If an attacker is able to intercept traffic behind the proxies, and present a valid certificate for the update domain, the attacker will be able to read and modify update traffic. This traffic is used to provide live updates to the client, but potential exploitation is reduced by cryptographic signatures of binary updates.
Recommendation	Use pinned certificates for all sensitive servers.

Appendix A: Vulnerability Field Definitions

The following sections describe 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: Lantern client detection Proof of Concept

The following HTML file uses Javascript/jquery to detect whether Lantern is running as described in finding NCC-LANT16-001 on page 10. It is adapted from a solution here: https://stackoverflow.com/questions/8937158/how-to-check-whether-a-port-is-open-at-clients-network-firewall

```
    <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose
        .dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<script
        src="http://code.jquery.com/jquery-1.12.4.min.js"
        integrity="sha256-ZosEbRLbNQzLpnKIkEdrPv7lOy9C27hHQ+Xp8a4MxAQ="
        crossorigin="anonymous"></script>
        <!--<script type="text/javascript" src="jquery-1.7.2-min.js"></script>i-->
    </head>
    <body>
        <script type"text/javascript">
            var isAccessible = null;
            function checkConnection() {
                var url = "http://127.0.0.1:16823/proxy_on.pac?1468350415635649072" ;
                $.ajax({
                    url: url,
                    type: "get",
                    cache: false,
                    dataType: 'jsonp', // required for cross domain support
                    crossDomain : true,
                    asynchronous : false,
                    jsonpCallback: 'deadCode',
                    timeout : 1500, // set a timeout in milliseconds
                    complete : function(xhr, responseText, thrownError) {
                        if(xhr.status == "200") {
                           isAccessible = true;
                           $("#msgid").html("Lantern found: "+isAccessible);
                        }
                        else {
                           isAccessible = false;
                           $("#msgid").html("Lantern found: "+isAccessible);
                        }
                    }
                });
            }
            $(document).ready( function() {
                checkConnection();
            });
        </script>

  <div id="msgid">
  </div>
    </body>
</html>
```