PREPARED FOR
Barath Raghavan, De Novo Group

PREPARED BY
Hans Nielsen, Matasano
Nikki Brandt, Matasano

RANGZEN SECURITY ASSESSMENT REPORT

Friday, May 29, 2015

matasanosecurity
part of nccgroup

info@matasano.com
(888) 677-0666

48 W 25th St
4th Floor
New York, NY
10010

55 E Jackson
Suite 450
Chicago, IL
60604

111 W Evelyn Ave
Suite 101
Sunnyvale, CA
94086

executive summary

SYNOPSIS

During May 2015, De Novo Group engaged Matasano Security to conduct a security assessment of the Rangzen Android application. The application provides a distributed anonymous messaging system, allowing for communication when centralized network services have been disabled. This assessment was open ended and time boxed. Source code was provided, and the entire application was in scope. The cryptography portions of Rangzen are covered by another assessment.

The key findings: The assessment uncovered a variety of flaws including:

- A vulnerability in peer detection allowing an attacker to efficiently prevent any user in a wide area from connecting to other real users, stopping all communications.
- Multiple components of the paper were either partially or not implemented, leaving Rangzen in a compromised state.
- Several issues related to secure handling of stored data allow for easier compromise of a given user.

SCOPE

Matasano's evaluation included the full Rangzen Android application. Testing was performed using Matasano test devices running the Rangzen code, simulating a real Rangzen deployment.

Components examined include all wireless communications, the anonymity guarantees, and features mentioned in the Rangzen paper. Common Android issues, such as secure file storage, network communication, and library issues were also tested. The cryptography was covered by a separate assessment.

LIMITATIONS

Many components of the Rangzen paper were not implemented in the application at the time of the engagement. For example, message trust decay, known-peer prioritization, deleting messages, and unfriending users were partially or not implemented. Vulnerabilities that were a result of an unimplemented feature are called out in the report.

methodology

SYNOPSIS

This project consisted of source-assisted penetration testing of the Rangzen application. In a source-assisted penetration test, consultants are given full access to an application, and also the application source code. This allows the consultants to use both black-box and white-box testing approaches to maximize coverage and productivity. The purpose of these tests was to bypass the security of the Rangzen application, and document each resulting security finding.

TEAM

From Matasano: the testing team consisted of Hans Nielsen and Nikki Brandt.

From De Novo Group: the project was managed by Barath Raghavan, and the technical point of contact was Adam Lerner.

TEST PLAN

For the Rangzen Android application, our test plan was as follows.

- **Implementation vulnerabilities:** The entire application was tested to ensure that an attacker would not be able to perform an action that could result in a business logic bypass or otherwise leak information about users of the application.
- **Secure Communication:** Matasano examined all network communications made by the application, checking that they use secure transports.
- **Secure Storage:** The storage of secrets in the Android application was assessed for common issues, such as weak master keys, storage in insecure locations, and the ability to back up sensitive data.
- **Intents:** All intents for the application were checked to ensure that no out-of-sequence actions could be initiated, and that no user data could be either modified or retrieved by other applications on the system.
- **Native code:** Any native code (i.e. JNI) discovered in the application was examined for potential memory corruption issues.
- **WiFi Direct vulnerabilities:** Matasano tested the application for scenarios where an attacker can use the WiFi Direct connection to disrupt Rangzen.
- **Bluetooth vulnerabilities:** the testers checked for scenarios where an attacker can sniff or man-in-the-middle Bluetooth connections to subvert Rangzen.
- **Anonymity compromise:** Matasano investigated scenarios in which an attacker can reveal the identity of other nodes in the network and/or the author of specific messages.
- **Priority tampering:** tested priority tampering to see whether or not an attacker can circumvent Rangzen's trust system to promote malicious posts.
- **Vulnerabilities related to third-party libraries (Wire and ZXing):** testers examined use of third-party libraries to determine whether they exposed Rangzen to additional attacks, such as data handling issues with Protocol Buffers or anything that could lead to memory corruption.

vulnerability summary

The table on the next page lists all of the issues discovered during the project. It is to be used by both Matasano and De Novo Group to ensure that all vulnerabilities are successfully managed. The definitions of the status columns are described below.

RISK SCALE

Matasano 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 Matasano'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.

- **Critical:** Implies an immediate, easily accessible threat of total compromise. Remediation should be pursued with the utmost speed.
- **High:** Implies an immediate threat of system compromise, or an easily accessible threat of large-scale breach. Remediation should be pursued quickly.
- **Medium:** A difficult to exploit threat of large-scale breach, or easy compromise of a small portion of the application. Remediation should be aggressively scheduled, but need not disrupt other urgent matters.
- **Low:** Implies a relatively minor threat to the application. Future plans should be made to remediate the issue.
- **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.

STATUS DEFINITIONS

- **Reported:** Matasano has found what we believe to be a vulnerability. At the time of this version of this document, De Novo Group is yet unaware of this vulnerability.
- **Confirmed:** De Novo Group confirmed the existence of the vulnerability and consensus on the risk has been reached.
- **False Positive:** The vulnerability discovered by Matasano is mitigated or does not exist.
- **Awaiting Retest:** The vulnerability has been fixed by De Novo Group and requires validation from Matasano.
- **Fixed:** Both parties (Matasano and De Novo Group) agree that the vulnerability has been fixed and/or mitigated.

vulnerability summary

VULNERABILITY CHART

#	ISSUE	SEVERITY	STATUS
1	**Compromised nodes can't be removed from the network**	Low	Reported
2	**Application doesn't require authorization before sending messages**	Low	Reported
3	**Maximum number of exchanged messages not enforced**	Low	Reported
4	**Rangzen private keys stored insecurely**	Low	Reported
5	**Application can be backed up**	Low	Reported
6	**An attacker can send arbitrarily long messages**	Low	Reported
7	**No limit on the number of peers added from WiFi Direct detection**	Medium	Reported
8	**Message priorities may reveal friendship connections between nodes**	Low	Reported
9	**An attacker can artificially boost priority of messages**	Low	Reported

vulnerability tracking details

#1	COMPROMISED NODES CAN'T BE REMOVED FROM THE NETWORK
Status	Reported
Severity	LOW
Impact	A compromised node can continue to spread disinformation unchecked even if the compromise is detected. Lack of unfriending ability and lack of trust decay mean a compromised node may stay active even after compromise is detected
Details	A trusted node could be compromised at any time. One likely route of compromise would be via theft or capture of the trusted device. Once compromised, the attacker could use the device to continue to send messages. These messages would appear to other nodes to be completely legitimate, and indistinguishable from messages coming from other trusted parties. The root of this problem lies in the fact that trust does not ever decay and cannot be forcibly revoked, as there is no unfriending ability in the application. Even if the legitimate owner of the device detected the theft, they (and their network of trust) would have no recourse.
Recommendation	Allow removal of existing trusted friends, although this is complicated as friends do not have a mapping from name to ID. Consider making trust partially decay over time, requiring contact with other users to restore full trust.

#2	APPLICATION DOESN'T REQUIRE AUTHORIZATION BEFORE SENDING MESSAGES
Status	Reported
Severity	LOW
Impact	An attacker with (even brief) physical access to a trusted device can post disinformation that will be spread with that device's trust level.

Details	There is no authorization check before messages are posted on Rangzen. If an attacker is able to achieve physical access to a device, even for a brief period, they can easily post messages from that device. The messages sent from the device will go out with the trust level associated with the device. Due to the anonymous nature of Rangzen, not even the person whose device is compromised in this attack will know that the message was sent by an attacker.
Recommendation	One possible countermeasure here is to require users to enter a PIN to post a new message is posted. If multiple incorrect PIN entries are detected, the application can wipe the secret keys to prevent an attacker from obtaining them. Another option is to force users to have a passcode or lock pattern enabled. By reading the value of LOCK_PATTERN_ENABLED in android.provider.Settings.Secure, Rangzen can determine whether a user's phone is protected.

#3	MAXIMUM NUMBER OF EXCHANGED MESSAGES NOT ENFORCED
Status	Reported
Location	java/org/denovogroup/rangzen/Exchange.java (line 81)
Severity	LOW
Impact	An attacker can monopolize the bandwidth of a legitimate node by sending a flood of messages above the intended exchange threshold.
Details	As designed, only 100 messages should be exchanged in each Rangzen transaction. However, the number of messages exchanged is only checked on the sender side. The overall amount of data exchanged is capped at 1MB per exchange, and appropriately checked on both ends of the transaction. Since the software is intended to be open source, it is plausible that an attacker can make a minor modification to the code in order to send an arbitrarily large number of messages. They can also then programmatically generate a corresponding number of messages in their local message store. After establishing this base, when the attacker comes into contact with an innocent node, the innocent node will be flooded with the spam messages. This may cause the innocent node to miss legitimate updates from other nodes in the area, and may also cause unexpected UI problems within the Rangzen application itself.

Recommendation	Check the number of messages exchanged on the receiver's end. Either close the connection after 100 messages have been exchanged, or drop additional messages instead of adding them to the message store.

#4	RANGZEN PRIVATE KEYS STORED INSECURELY
Status	Reported
Location	org/denovogroup/rangzen/FriendStore.java (line 225)
Severity	LOW
Impact	An attacker with physical access to the device or access to a user's phone backup could steal the Rangzen private key and use it to masquerade as the phone's owner in future communications.
Details	The public / private keypair for Rangzen, which is used as the user's anonymous identity in the Rangzen network, is stored as plaintext in RangzenData.xml. This file is located in the shared_preferences directory for the application. This directory is accessible if the phone has been rooted, or if a backup is made of Rangzen. If an attacker obtains brief physical access to a rooted device, they can pull RangzenData.xml off of the phone using a few simple adb commands.
Steps To Reproduce	1. Load Rangzen onto a rooted test device. 2. Connect to the device and browse to /data/data/org.denovogroup.com/rangzen. 3. Open RangzenData.xml and note that it contains both the public and private key in plaintext, encoded as base64 strings.
Recommendation	Use the Android Keystore Provider to store keys. Private keys can be stored either in software or in hardware (on supported devices), and are only exposed to the application through an interface that allows signing and encryption, preventing accidental key leakage. Note that on devices with software key storage, keys can be recovered on a rooted device. See https://developer.android.com/training/articles/keystore.html for more information.

#5	APPLICATION CAN BE BACKED UP
Status	Reported
Location	ui/Rangzen/AndroidManifest.xml (line 22)
Severity	LOW

Impact	An attacker finds a backup of Rangzen's data on a user's computer. They use the contained keys and friends to deanonymize and impersonate the user.
Details	The application manifest currently specifies that Rangzen can be backed up by a user. Allowing a user to backup Rangzen data exposes them to further attacks against their computer in addition to their phone.
Recommendation	Set android:allowBackup="false" on the application element in the AndroidManifest.xml to disallow all application backups. See http://developer.android.com/guide/topics/manifest/application-element.html#allowbackup for more details.

#6	**AN ATTACKER CAN SEND ARBITRARILY LONG MESSAGES**
Status	Reported
Location	ui/Rangzen/res/layout/makepost.xml (line 45) java/org/denovogroup/rangzen/MessageStore.java (line 206)
Severity	LOW
Impact	An attacker can drown out messages from anyone with a trust level equal to or below theirs by crafting extremely long messages.

Details	It is possible for an attacker to send a message of an arbitrary length by hardcoding a message in the app and rebuilding it. This is achievable because the only check on individual message length is built into the UI. The TextView field is limited to 140 characters, so an attacker cannot send an arbitrarily long message through the normal means; however, since the software is open source, it is plausible for the attacker to download the source, hardcode a disruptively long message or set of messages, and rebuild the app, thereby circumventing the check entirely. Due to the small form factor of the mobile devices Rangzen is intended for, the presence of overly long messages severely hampers the usability of the application. By also exploiting knowledge of the organization method (messages are sorted first by priority, then sorted alphabetically), an attacker can use overly long inputs to effectively drown out any messages at or below their own trust level. Additionally, we have observed that overly long messages cause UI problems long before the maximum message size is reached. So at present, this feature could also be used to render Rangzen totally unusable on a victim's device, by merit of causing the device to crash every time the UI is accessed.

Steps To Reproduce	1. Insert the following code into Opener.java around line 130 (after creation of the MessageStore). Note that this code could be also be added through lower-level means such as smali / baksmali.
	
	`char[] somebytes;`
	`somebytes = new char[522244];`
	`while(i < 522242)`
	`{`
	`  somebytes[i] = '*';`
	`  somebytes[i+1] = 'o';`
	`  i = i + ;`
	`}`
	`String somestring;`
	`somestring = new String(somebytes);`
	`messageStore.addMessage(somestring, 1L);`
	
	1. Rebuild experimentalApp and reinstall it on the test device.
	2. Open the UI, wait for it to load, and observe that the overly long message appears. Note that it may crash if you attempt to interact with it before the message appears.
	3. Bring another device running Rangzen into proximity and observe that the overly long message is successfully received on that device as well. Notice that any messages with the same (or lower) priority are extremely difficult to see.

Recommendation	Check the message length before storing it in the message store. Reject any messages over a certain length. Ideally, the maximum length stored will correspond with the maximum length accepted by the UI.

#7	NO LIMIT ON THE NUMBER OF PEERS ADDED FROM WIFI DIRECT DETECTION
Status	Reported
Severity	MEDIUM
Impact	An attacker can spoof Wifi Direct device names, causing the Rangzen app to try and pair with a huge list of fake peers, preventing it from communicating with the rest of the network.
Details	The Wifi Direct peer discovery process adds all potential matches discovered to the list of peers. To pose as a potential match, an attacker has to have a device name of the form "RANGZEN-11:22:33:44:55:66", where the sequence of numbers is the device's Bluetooth MAC address. There is no filtering or rate-limiting of these devices on the client-side. By flooding a client with many spurious peers, an attacker can make it extremely unlikely for a client to connect to a real peer.
Recommendation	Prioritize existing known-good peers over new peers. Cap the number of discovered peers. When unable to connect to a peer, immediately search for another peer, rather than delaying the search process further. When saving peers, consider building a Bloom filter of known-working peers to prevent leaking which peers are known by a given device.

#8	MESSAGE PRIORITIES MAY REVEAL FRIENDSHIP CONNECTIONS BETWEEN NODES
Status	Reported
Severity	LOW
Impact	An attacker can identify nodes by mapping their friendship graphs using message priorities.

Details	An attacker may be able to capture the priorities going both directions and use those priorities to determine the network of friends connecting the two users. This may be used to reveal the identity of any given node. Priorities can be collected either via an attacker who is actively participating in Rangzen, or by an attacker who is eavesdropping but not using the Rangzen app. This vulnerability exists because neither the noise factor proposed in the Rangzen white paper nor per-message encryption have been implemented at present.
Recommendation	Add a noise factor to the message priorities to obscure friendships further as proposed in the Rangzen white paper. Also consider encrypting messages to prevent non-participating, eavesdropping nodes from collecting the priority data.

#9	**AN ATTACKER CAN ARTIFICIALLY BOOST PRIORITY OF MESSAGES**
Status	Reported
Severity	LOW
Impact	An attacker can increase or decrease the priority of a message before sending it onward to legitimate nodes.

Details	Due to the open source nature of the project, an attacker can manually insert priorities into messages programmatically. The insertion of attacker-selected priorities can result in boosting or drowning out important messages on the receiver's end. Currently, the application will check the priority assigned to a message on the receiver's end and assure that it is between -1.0 and 2.0. An attacker can choose any value in that range to apply to a message, however, whether or not the message originated with the attacker. The problem arises because the receiver has no way of knowing what the original priority was. The attacker-chosen priority will be used in calculating the new display priority instead. The boosting attack works as long as the receiver trusts the attacker even slightly. If the receiver and the attacker have at least one friend in common, the message's priority can be boosted as high as 199 priority (higher even than a self-post). The extent of the impact of the tampering scales with the size of the legitimate node's network and the number of friends the legitimate node has in common with the attacker. For a legitimate node that has only 1 friend, who is also a friend of the attacker, the impact of the tampering is large; the attacker can send messages with a priority up to 199. When the legitimate node has a larger network with few shared friends, the attack's impact is lessened; however, it may still be used to raise the attacker's messages relative to other legitimate messages. The downvoting attack (priority decrease) works only in a scenario where the legitimate receiver has never seen the message the attacker is sending before. This limitation is due to the fact that Rangzen stores the maximum priority seen for a given message. If the receiver has already seen the message with a higher priority, Rangzen will ignore the attacker's new, lower priority; however, if the receiver has not seen the message previously, the receiver will accept the lowered priority. This could lead to the legitimate node missing truly high priority messages. This has potential to be devastating when combined with other attacks such as the Wi-Fi direct peer suppression attack.

Steps To Reproduce	1. Obtain two devices and have them each add a third device (a fake QR code is fine) as a friend. . Modify getPriority's return (MessageStore.java, line 251) to this: return 1.99; Rebuild the source and install the resulting app on one device. This device now represents an attacker. . Force a refresh by doing something like posting a new message. Observe that all messages on the attacker device will now have priority 199. Observe that the second device should now receive any messages posted by the attacker with priority 199 as well.
Recommendation	Without compromising anonymity, this problem does not have a clear solution.

appendix a: android secure file storage

Android provides a minimal set of facilities for secure file storage. Beyond the KeyStore Provider, introduced in 4.3, an application has to implement its own system to keep data safe. To provide a reasonable level of security, an application must protect the data with a PIN. This PIN is used to derive encryption and message authentication keys.

KEYSTORE PROVIDER

The KeyStore Provider[1] was introduced in Android 4.3. The KeyStore allows storage of secrets to be handled by the operating system rather than the application. This data won't be saved in backups of the application. In addition to raw data, KeyStore can also handle data such as private keys and perform signing operations without ever providing the key to the application.

The system can perform secret storage in software or in hardware. There is no guarantee a given device will support hardware storage. That said, storing sensitive data in the KeyStore is in all cases a win over storing it as a file in the user data area, as it is managed by the operating system and not vulnerable to attacks such as path traversal.

IMPLEMENTING SECURE STORAGE

Users will need to have a PIN to access the application data, as the system provides little in the way of hardware credential storage. This PIN should be used in combination with a key derivation function, such as `scrypt`, `bcrypt`, or `PBKDF2`, in order of desirability. All of these key derivation functions are tunable, allowing the developer to determine how hard and how long it takes to complete the derivation. The amount of time should make it prohibitive for an attacker to launch a large-scale brute-force attack against the encrypted files.

Note that PINs tend to be short, and often consist of only four digits. If the PIN is this short any tuning factor high enough to prevent brute-forcing will also make the application unusably slow. Users should be encouraged to choose a longer PIN if possible.

For actually encrypting the data, consider using a library such as Facebook's Conceal[2]. To provide the derived keys to the library, implement the `com.facebook.crypto.keychain.KeyChain` interface. If that is not possible, encrypt the data with AES-256-CBC, generating a new random IV on each encryption. Ensure the SecureRandom class is used to generate random numbers. The encrypted data must be authenticated with HMAC-256, and needs to be verified before decryption.

---

1 https://developer.android.com/training/articles/keystore.html

2 https://github.com/facebook/conceal