Video URL: https://youtu.be/PTesSzCtHc4

Proposed New Feature of Bitcoin Core

By: The Foobar Fighters

Roles and Responsibilities

LEADER: Makayla Mcmullin → Implementation #2: New Vault Module, Feature Diagrams, SAAM

PRESENTER: Daniel Dickson → Quality Attributes, Lessons & Limitations, slides

PRESENTER: Aniket Mukherjee → Enhancement Overview, Potential Risk, slides

Maia Domingues → Implementation #1: Spreading the Functionality, Testing Plans, SAAM

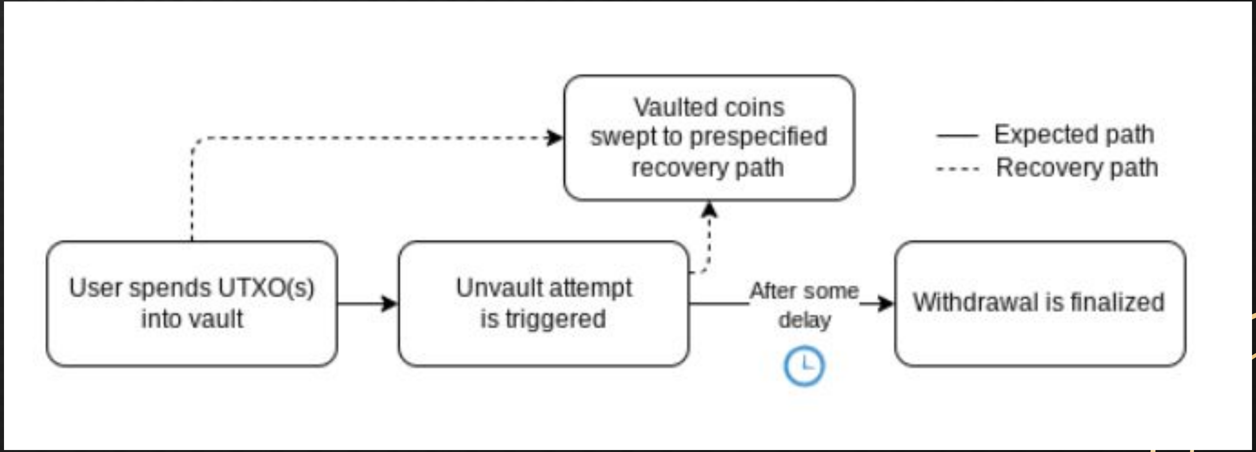
Devon Gough → Abstract, Introduction, LaTeX

Lucas Patoine → Use Cases & Conclusion

EVERYONE: Data Dictionary, General Research, Naming Conventions

Our Proposed Enhancement

- Proposed Feature: “Vault” to prevent fraudulent transactions
    - Bitcoin Core is currently lacking in terms of countermeasures for when someone tries to steal another user’s bitcoins
- User would be alerted to the attempted theft
- Bitcoin would temporarily be diverted to a more secure wallet (vault)
- To open vault, user must broadcast two separate transactions in two separate blocks
    - Aims to stop user key from being compromised

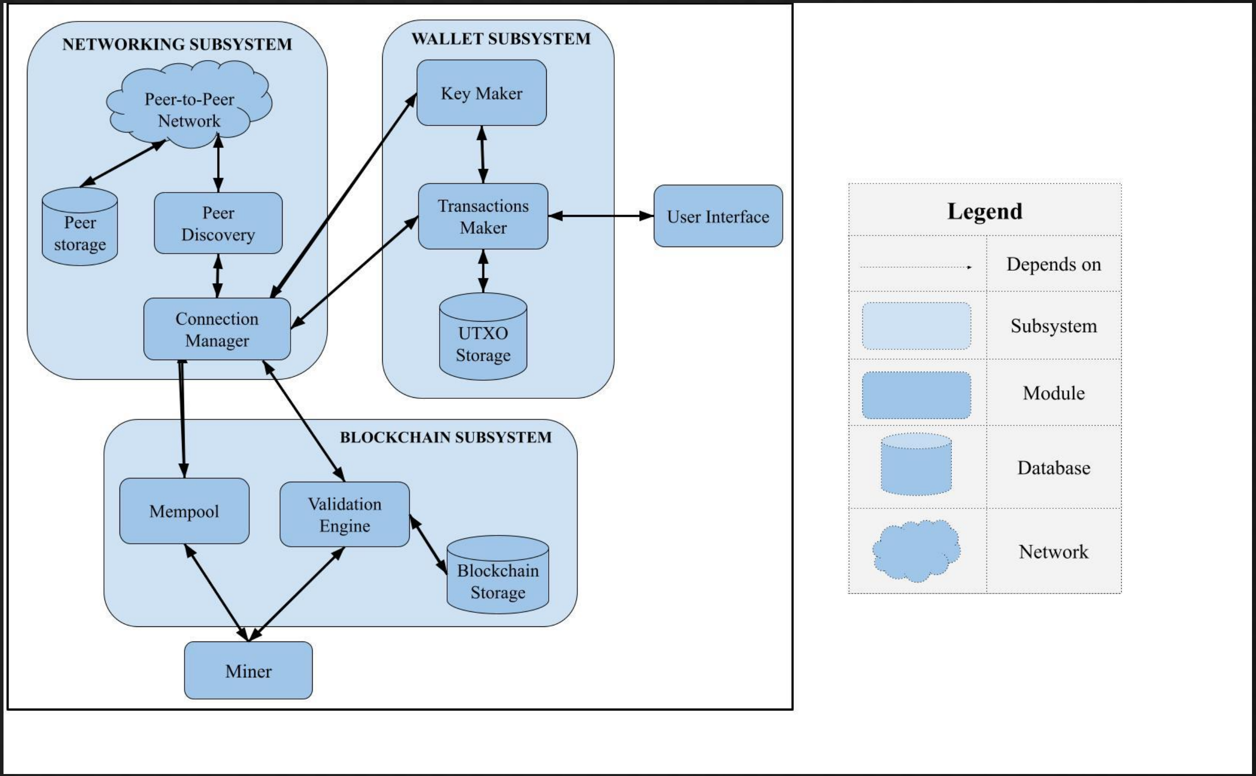

Vaulted coins swept to prespecified recovery path
Expected path
Recovery path
User spends UTXO(s) into vault
Unvault attempt is triggered
After some delay
Withdrawal is finalized

Implementation Method #1: CheckTemplateVerify

- Implement functionality over pre-existing modules within the software architecture
  - Change each of the current modules' function; implement CTV to create vault-like structure

- Add new operation code to allow verification and enforcement of security features & recovery failsafes
  - Requiring multiple signatures for a withdrawal, time lock, template of future transaction hash serials to reference, etc.
- Modify existing wallet & blockchain modules to add vault support
- When a user wishes to spend funds stored in their vault, CTV should engage and only allow spending under defined specific circumstances
- Multiple UTXOs can be consolidated into a single output, which reduces its set size

Implementation Method #1: Diagram



Implementation Method #1: Drawbacks

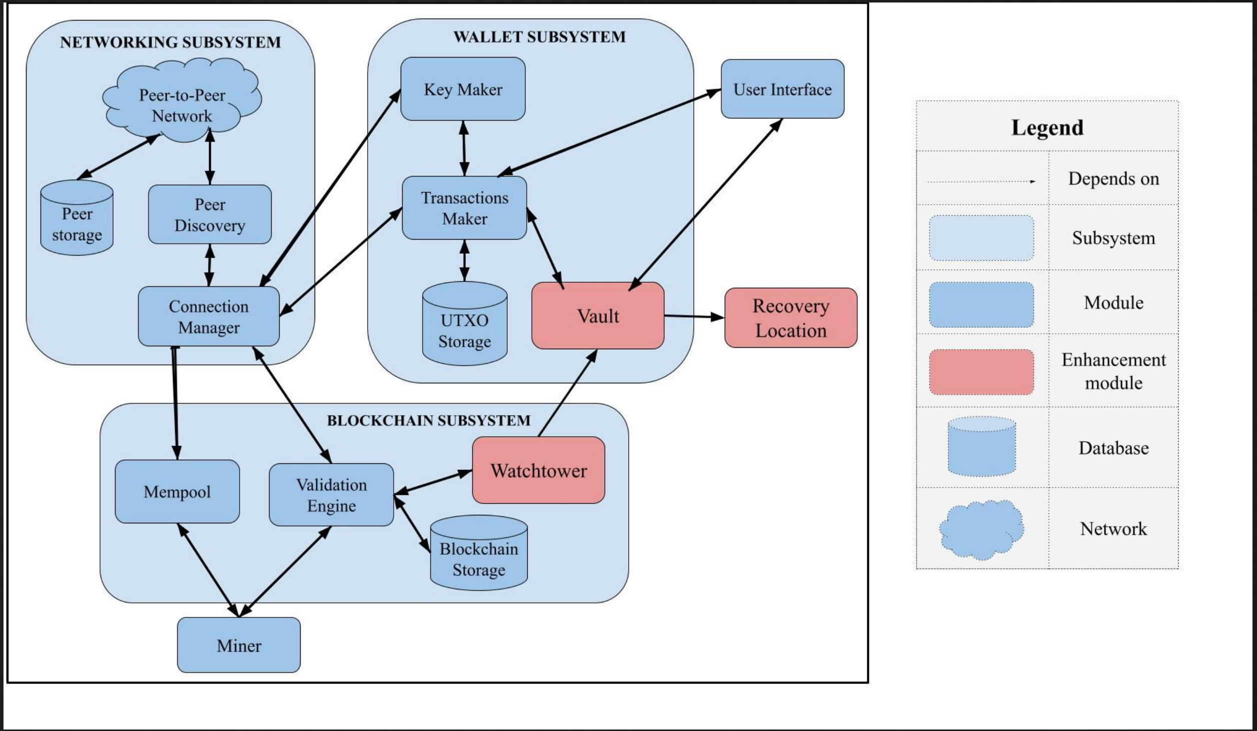
- Less flexibility than a singular module, more points of failure
  - Harder for developers to implement changes
- The CTV code would be visible on the blockchain, and thus to all Bitcoin Core users
  - Ironically, this could also be a major security risk if not mitigated properly

Implementation Method #2: New Modules

- Implement functionality using additional modules
    - Allows other pre-existing modules to remain unchanged, better for testing & development
    - Better modularity of system, no change to system architecture structure

- Three new modules: Vault, Recovery Location, and Watchtower
    - VAULT - receives UTXOs from Transaction Maker & stores them
    - WATCHTOWER - monitors blocks passing through the Validation Engine for any UTXOs in Vault
        - If it sees a UTXO in blockchain that's also in Vault, it sends an un-vaulting attempt notification to the Vault, which then sends notification to UI for user approval
    - RECOVERY - If the UI notification was not approved, UTXOs are sent here, otherwise they're sent to Transaction Maker

Implementation Method #2: Diagram



SAAM

- There are two relevant methods of implementation for a Vault enhancement on the Bitcoin Core system: modifying pre-existing modules to perform the enhancement or implementing the enhancement using new modules. We will discuss both options in detail then analyse them using the SAAM method to determine which implementation is more satisfactory to the stakeholdersc
- The users' most important non-functional requirements for the enhancement are security, usability, performance, compatibility and portability. Security is important to the users because the system holds all of their currency
- Usability is important to the users because they want to easily navigate the system, conveniently manage their cryptocurrency without errors and for the system to run efficiently on their host system
- Performance is important to users because they will want to make transactions quickly without unnecessary lag
- Compatibility and portability are important to users because they will require the system to run smoothly across multiple environments and work well with pre-existing systems on their host computer

Stakeholders: Users

- Security
    - Option 2 provides better security, depending on strength of recovery system
    - Option 1 leaves security vulnerable since the opcode script is publicly accessible over the blockchain
- Usability
    - Option 1 may lower learnability, Option 2 may lower efficiency
- Performance
    - Option 1 lowers overall performance due to increased time needed for processing, Option 2 may limit throughput due to time spent waiting on signals/work from new dependencies
- Compatibility/Portability
    - Option 1 changes overall codebase, may cause compatibility issues
    - Option 2 allows enhancement to be easily downloaded and installed due to no changes in existing components

Stakeholders: Developers

- Maintainability
  - Option 2 is better, as all the code is in one place instead of being scattered throughout
- Testability
  - Option 2 is better, easier to test separate modules than update tests to existing modules to cover newly implemented files/codeblocks
- Extensibility
  - Option 2 is better, easier to keep track of existing changes and easier to upgrade modules separately from the rest of the software

Effects on Quality Attributes

- Maintainability: Additional components in system = DECREASED
- Testability: Amount of testing increases, but tests should still be effective and straightforward to implement
- Performance: Slowed down due to extra step in process, but ability to function properly is improved (increased security)
- Evolvability: Keeping everything new to one module makes it easier to keep track of where all the components of the system are, making it easier to change them going forward as necessary

Testing Plans

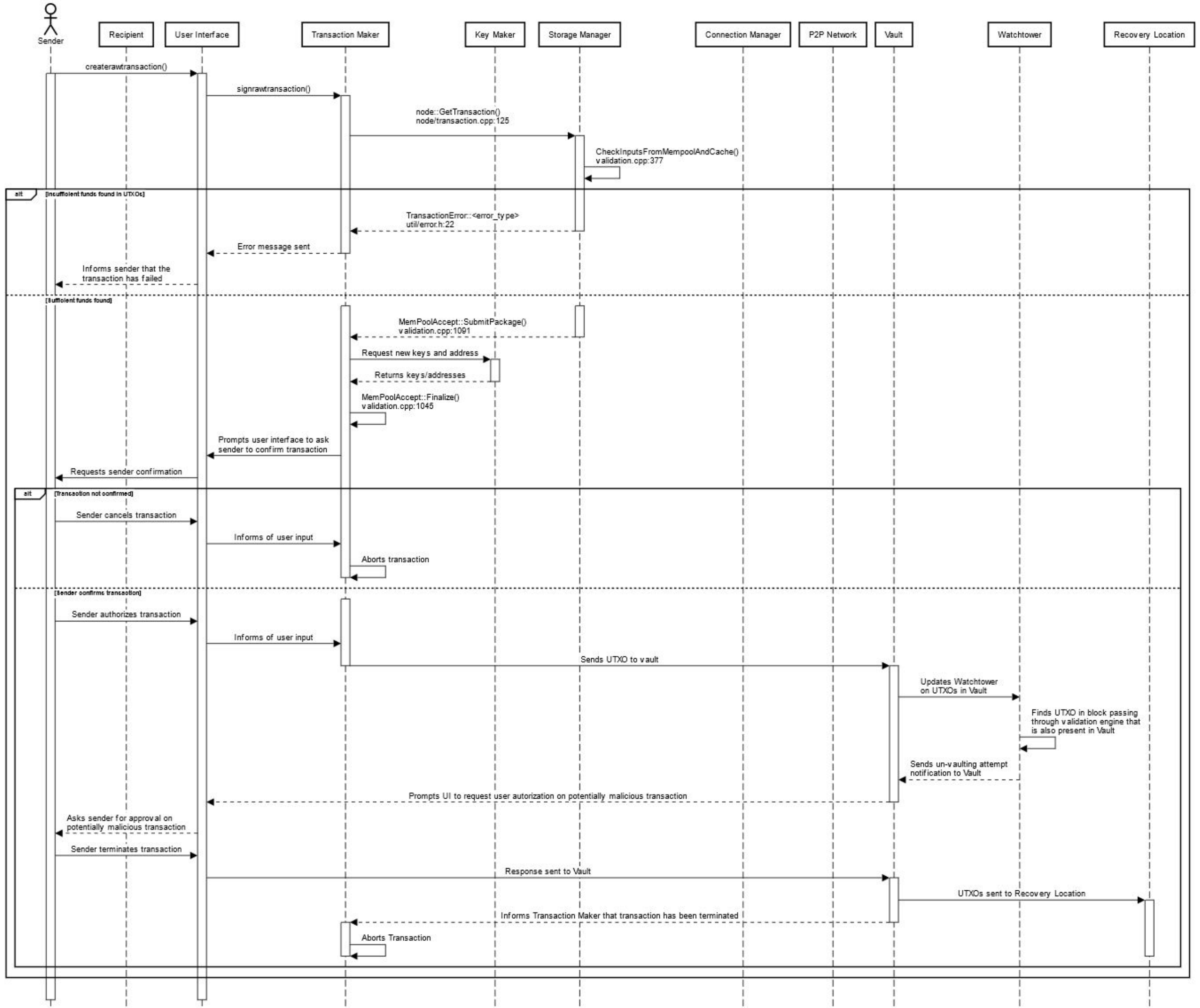
- The interaction between all pre-existing modules and the new proposed vault functionality must be taken into account during the testing phase of development
- For example, the functionality of the current system before any changes take effect may not be fully compatible with the proposed framework of a vault system, and modules will most likely have to be edited, possibly substantially, to account for these new features
- For implementation 1: edit the existing tests of the modules being changed. This will mostly happen with unit testing, where each function will have its boundaries in place before the code is finished (ideally, before the code is even started).
- For implementation 2: The feature is implemented as its own module, leaving all the existing code and tests alone. This allows more concise and simple files to be written all in one place, decreasing the chance that some functionality slips through untested due to programmer error.

Potential Risk

- Due to the extra security measures that a vault would require(the manual confirmation of any and all transactions from one wallet to another), it would slow the speed of many transactions that Bitcoin Core handles, as it would no longer automatically go through
- Speed is also reduced further in the event of an actual breach where funds are diverted to an alternate wallet
- Furthermore, there's also the fact that the alternative wallet is also compromised, which means that even if the user chooses to sweep their coins into another wallet to protect them, it might only be a temporary security solution to their initial wallet being compromised, leaving their coins vulnerable to theft yet again

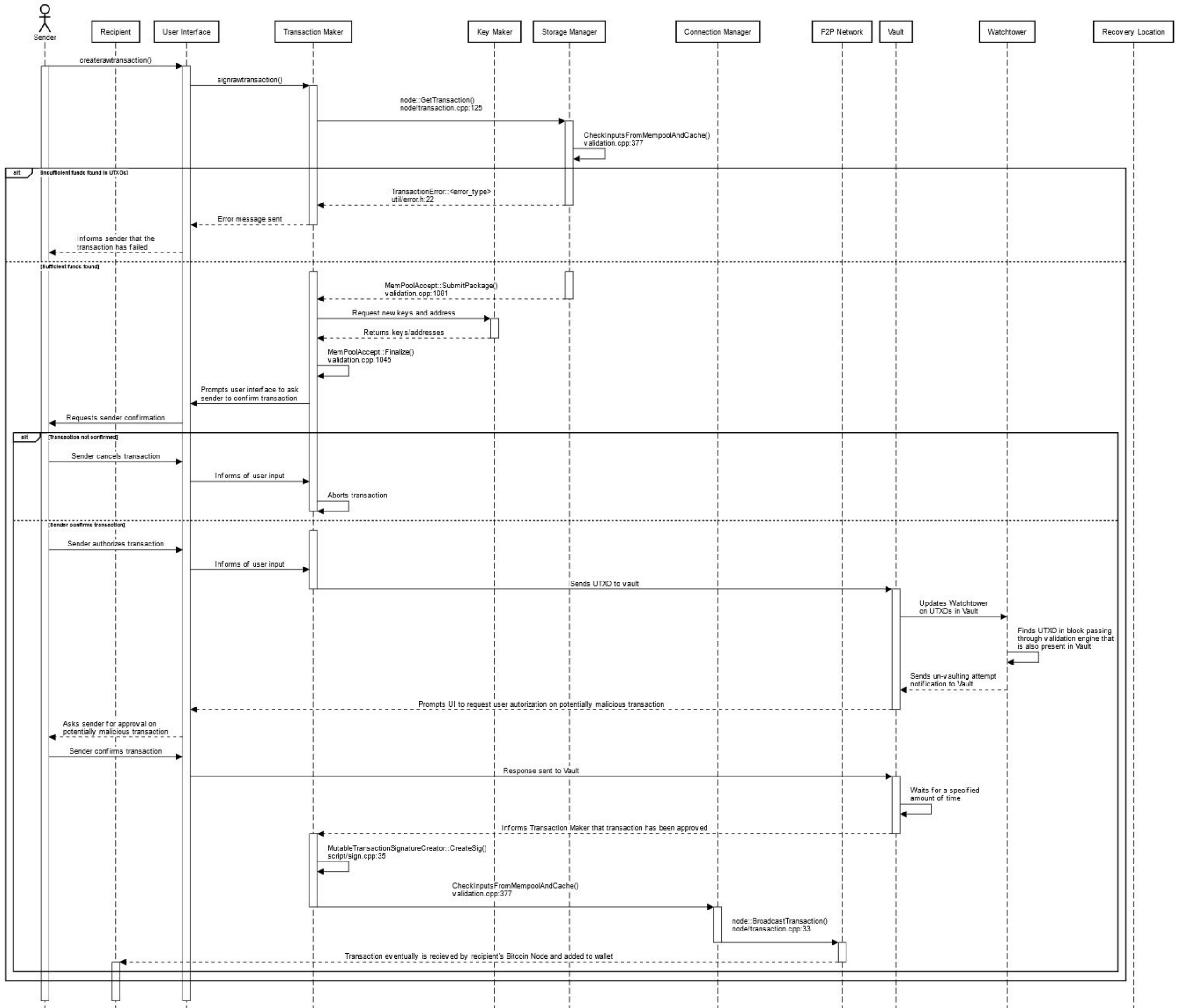
Use Case 1

- A fraudulent transaction is made, the user is informed, and proceeds to terminate it


Sender
Recipient
User Interface
Transaction Maker
Key Maker
Storage Manager
Connection Manager
P2P Network
Vault
Watchtower
Recovery Location
createrawtransaction()
signrawtransaction()
node::GetTransaction() node/transaction.cpp:125
CheckInputsFromMempoolAndCache() validation.cpp:377
alt
[Insufficient funds found in UTXOs]
TransactionError::<error_type> util/error.h:22
Error message sent
Informs sender that the transaction has failed
[Sufficient funds found]
MemPoolAccept::SubmitPackage() validation.cpp:1091
Request new keys and address
Returns keys/addresses
MemPoolAccept::Finalize() validation.cpp:1045
Prompts user interface to ask sender to confirm transaction
Requests sender confirmation
alt
[Transaction not confirmed]
Sender cancels transaction
Informs of user input
Aborts transaction
[Sender confirms transaction]
Sender authorizes transaction
Informs of user input
Sends UTXO to vault
Updates Watchtower on UTXOs in Vault
Finds UTXO in block passing through validation engine that is also present in Vault
Sends un-vaulting attempt notification to Vault
Prompts UI to request user autorization on potentially malicious transaction
Asks sender for approval on potentially malicious transaction
Sender terminates transaction
Response sent to Vault
UTXOs sent to Recovery Location
Informs Transaction Maker that transaction has been terminated
Aborts Transaction

Use Case 2

- The user makes a transaction, is alerted, and approves of it.



Sender
Recipient
User Interface
Transaction Maker
Key Maker
Storage Manager
Connection Manager
P2P Network
Vault
Watchtower
Recovery Location
createrawtransaction()
signrawtransaction()
node::GetTransaction() node/transaction.cpp:125
CheckInputsFromMempoolAndCache() validation.cpp:377
alt
[insufficient funds found in UTXOs]
TransactionError::<error_type> util/error.h:22
Error message sent
Informs sender that the transaction has failed
[sufficient funds found]
MemPoolAccept::SubmitPackage() validation.cpp:1091
Request new keys and address
Returns keys/addresses
MemPoolAccept::Finalize() validation.cpp:1045
Prompts user interface to ask sender to confirm transaction
Requests sender confirmation
alt
[transaction not confirmed]
Sender cancels transaction
Informs of user input
Aborts transaction
[sender confirms transaction]
Sender authorizes transaction
Informs of user input
Sends UTXO to vault
Updates Watchtower on UTXOs in Vault
Finds UTXO in block passing through validation engine that is also present in Vault
Sends un-vaulting attempt notification to Vault
Prompts UI to request user autorization on potentially malicious transaction
Asks sender for approval on potentially malicious transaction
Sender confirms transaction
Response sent to Vault
Waits for a specified amount of time
Informs Transaction Maker that transaction has been approved
MutableTransactionSignatureCreator::CreateSig() script/sign.cpp:35
CheckInputsFromMempoolAndCache() validation.cpp:377
node::BroadcastTransaction() node/transaction.cpp:33
Transaction eventually is recieved by recipient's Bitcoin Node and added to wallet

Limitations & Lessons Learned

- Limited ability to research new feature due to:
    - Difficulty finding more information about up-to-date source code and developers' notes
    - Since this is a brand new feature we're suggesting, there was very little information online about how it could be added in
- We have far less experience working with Bitcoin Core than professionals who might be proposing additional enhancements
- Learned a lot about SAAM, software quality, and how fairly small changes in a system can have large effects on how it works and how it needs to be maintained
    - Different implementation styles can result in same overall effect with very different changes needed to be made
- This sort of thorough analysis is a valuable skill to be used going forward

Thank you for listening!