This was written in response to https://github.com/bitcoin/bitcoin/pull/27578 and any other number of occurrences of the topic.

“Standardness policy” is a term for a transaction that would be consensus-legal, but our node doesn’t want to relay for various reasons. Clearly if it’s consensus valid we want to accept it to not split the network if a miner chooses a “weird” transaction to mine.

There are N motivations for policy that I know of:

1) Anti-DoS: Only “cheap” things to validate get flooded to the network
2) Security: Certain types of transactions may mess up certain systems for no good reason
3) Upgrade hooks: We leave some things “forbidden” such that if we find a use for them later we don’t “confiscate” funds by refusing to relay e.g., a spend or pre-signed transaction
4) Pro-decentralization: Make it simple/cheap for miners to build blocks that pay well
5) Paternalism: We want wallet authors to use the least amount of public resources while still accomplishing their end goals(as long as that goal isn’t “attack the network”!).
6) Let people pay fees to make transactions in an acceptable API

All of these motivations obviously have to be weighed against the fact that bitcoiners want to make transactions, and miners want fees from those transactions. Ideally we make a mempool/relay system where both can live in relative harmony.

Let’s dive into the standardness restrictions circa Bitcoin Core 25

Three key places we’ll look if you’re following along at home, though there are a few more scattered about:

1) `IsStandardTx`
2) `AreInputsStandard`
3) `IsWitnessStandard`

All in `src/policy/policy.cpp`. These checks are turned off in testnet by default, and can be turned off in regtest optionally.

And now here’s the list of logical restrictions, and how I’m classifying them:

IsStandardTx

```
if (tx.nVersion > TX_MAX_STANDARD_VERSION || tx.nVersion < 1) {
    reason = "version";
    return false;
}
```

Only allow transaction nVersions of 1 or 2.

Labels: (3)
Example future uses: V3 policy https://github.com/bitcoin/bitcoin/pull/25038
Historically it was used to signal opt-in support for relative timelocks.

```
if (sz > MAX_STANDARD_TX_WEIGHT) {
    reason = "tx-size";
    return false;
}
```

Don't allow transactions that are too large
Labels: (1), (4)
Smaller transactions, especially pre-segwit, are cheaper to validate due to quadratic sighashing issues of legacy script. It also makes block creation easier with respect to the binpacking problem when you have smaller pieces to put in the bin, simplifying mining code/competitiveness.

```
if (txin.scriptSig.size() > MAX_STANDARD_SCRIPTSIG_SIZE) {
    reason = "scriptsig-size";
    return false;
}
```

Legacy scriptSig can't be bigger than "practical" at 15-of-15 multisig spend, the maximum possible pre-segwit due to P2SH limits.
Labels: (1)
Avoids signature bombs with quadratic hashing (pre-segwit issue). We'd like to softfork this out but can't because that's potentially theft.

```
if (!txin.scriptSig.IsPushOnly()) {
    reason = "scriptsig-not-pushonly";
    return false;
}
```

scriptSig can't have opcodes included.
Labels: (1), (2)
Recall that pre-segwit, all witness data went into the scriptSig, and pushing opcodes here was consensus-legal. Makes Bitcoin script even hard to reason about. We would like to softfork it out, but can't because that's potentially theft.

```
if (!::IsStandard(txout.scriptPubKey, max_datacarrier_bytes, whichType)) {
    reason = "scriptpubkey";
    return false;
}
```

Output scripts should match some "well known" pattern to be relayed.
Labels: (1), (2), (3), (4), (5)

This is a big one. It basically makes everything in the system easier to reason about. We know, for example, that no one can put a bare script transaction that consists of nothing but 20-of-20 bare multisigs to blow out the sigops limit in a block. It allows us to think about future patterns, like taproot outputs or other segwit versions, without having to make sure people aren't using said patterns already, therefore risking theft of user's funds. It's also a bit of paternalism, in that we think that anything "interesting" can likely be done within these script patterns, and want to encourage people to use those.

```
else if ((whichType == TxoutType::MULTISIG) && (!permit_bare_multisig)) {
    reason = "bare-multisig";
    return false;
```

Disallow relay of bare multisigs(allowed by default)
Labels: (1), (4), (5)
Please don't use bare multisigs. It makes Satoshi sad.

```
} else if (IsDust(txout, dust_relay_fee)) {
    reason = "dust";
    return false;
}
```

Don't make outputs that aren't likely to be spent because they're so small.
Labels: (1), (3), (4), (5)
Prior to something like utreexo being standard, every node has to carry every unspent TXO forever to stay in consensus with the network. We want to encourage outputs to be spent eventually. Maybe we can allow dust in near future circumstances provided we make sure they're swept ala Ephemeral Anchors: https://github.com/bitcoin/bitcoin/pull/26403

```
// only one OP_RETURN txout is permitted
if (nDataOut > 1) {
    reason = "multi-op-return";
    return false;
}
```

Only one OP_RETURN a transaction (and only up to 80 bytes of payload!)
Labels: (5)
This is one of the rare paternalism-only restrictions in my estimation. I'm for removing it, especially in a post-segwit world where witness data gets a deep discount for publishing random data.

AreInputsStandard

```
if (whichType == TxoutType::NONSTANDARD || whichType == TxoutType::WITNESS_UNKNOWN) {
    // WITNESS_UNKNOWN failures are typically also caught with a policy
    // flag in the script interpreter, but it can be helpful to catch
    // this type of NONSTANDARD transaction earlier in transaction
    // validation.
    return false;
```

Transaction inputs' scriptPubKey shouldn't be non-standard, or future segwit versions
Labels: (1), (2), (3)
Nonstandard scripts mean potentially scary legacy script behavior. Future segwit versions are left for future softforks, to make deployment easier.

```
CScript subscript(stack.back().begin(), stack.back().end());
if (subscript.GetSigOpCount(true) > MAX_P2SH_SIGOPS) {
    return false;
}
```

Legacy inputs can only have 15 signatures.
Labels: (1), (2), (4)
Legacy script sucks.

IsWitnessStandard

Now we're getting into modern script, which is a bit more interesting in some ways.

```
// Check P2WSH standard limits
if (witnessversion == 0 && witnessprogram.size() == WITNESS_V0_SCRIPTHASH_SIZE) {
    if (tx.vin[i].scriptWitness.stack.back().size() > MAX_STANDARD_P2WSH_SCRIPT_SIZE)
        return false;
    size_t sizeWitnessStack = tx.vin[i].scriptWitness.stack.size() - 1;
    if (sizeWitnessStack > MAX_STANDARD_P2WSH_STACK_ITEMS)
        return false;
    for (unsigned int j = 0; j < sizeWitnessStack; j++) {
        if (tx.vin[i].scriptWitness.stack[j].size() > MAX_STANDARD_P2WSH_STACK_ITEM_SIZE)
            return false;
    }
}
```

Segwitv0 P2WSH script size and stack item size limits.
Labels: (3), (5) ?
This I'm the most fuzzy on in some ways. These limits replicate some prior policy limits, so perhaps just precautionary principle here? Perhaps larger sizes end up being problematic in analysis, and can just be softforked out?

```
if (stack.size() >= 2 && !stack.back().empty() && stack.back()[0] == ANNEX_TAG) {
    // Annexes are nonstandard as long as no semantics are defined for them.
    return false;
}
```

Taproot annex must be empty
Labels: (2), (3), (7)

The annex is the most speculative of upgrade hooks in taproot it seems. There are a few ideas on how to use it, and a few issues to work around when it comes to coinjoin like scenarios. For example, did you know that taproot sighash currently doesn't cover *other* inputs' annex fields, allowing other inputs to inflate transaction size?
https://github.com/bitcoin-inquisition/bitcoin/pull/22

```
if ((control_block[0] & TAPROOT_LEAF_MASK) == TAPROOT_LEAF_TAPSCRIPT) {
    // Leaf version 0xc0 (aka Tapscript, see BIP 342)
    for (const auto& item : stack) {
        if (item.size() > MAX_STANDARD_TAPSCRIPT_STACK_ITEM_SIZE) return false;
    }
}
```

Tapscript should be BIP342 script
Labels: (3)
Straight forward upgrade hook, don't want people using it when it doesn't have any real consensus meaning yet.

Back to other areas of code

```
// Transactions smaller than 65 non-witness bytes are not relayed to mitigate CVE-2017-12842.
if (::GetSerializeSize(tx, PROTOCOL_VERSION | SERIALIZE_TRANSACTION_NO_WITNESS) < MIN_STANDARD_TX_NONWITNESS_SIZE)
    return state.Invalid(TxValidationResult::TX_NOT_STANDARD, "tx-size-small");
```

Don't allow transactions that are "really small" which can break merkle proofs for light clients.
Labels: (2), (5)
See CVE-2017-12842. The fact that we still disallow <64 bytes is a tiny bit of paternalism, in that there might be other things we have missed still, and the use-case for such isn't exceedingly strong so discussion was put on hold.

There are a number of things that are standardness or "standardness-like" but I didn't include but here are some with quick labels:

1) "Virtual transaction size" for transactions with lots of pre-taproot signature operations (1), (4), (5)
    a) Don't disallow relay of things like bare multisigs, but make them cost more to relay...
2) ancestor/descendant package limits (4)
    a) Computational limits of package feerate tracking/mining/etc
3) Script interpreter flags e.g., CLEANSTACK, SCRIPT_VERIFY_DISCOURAGE_UPGRADABLE_* (2), (3)
4) Sigops limits pre-taproot (1), (4) <--- no need to check sigops in tx when building block
5) ... and probably stuff I've forgotten

In my own personal estimation that only thing I can think of as pure paternalism(5) is OP_RETURN limits. Therefore in general I don't think there are easy answers and "just turn off standardness checks" is unlikely to ever be the right answer.