```
BIP: 12
Layer: Consensus (soft fork)
Title: OP_EVAL
Author: Gavin Andresen <gavinandresen@gmail.com>
Comments-Summary: No comments yet.
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0012
Status: Withdrawn
Type: Standards Track
Created: 2011-10-18
```

## Abstract

This BIP describes a new opcode (OP_EVAL) for the Bitcoin scripting system, and a new 'standard' transaction type that uses it to enables the receiver of bitcoins to specify the transaction type needed to re-spend them.

## Motivation

Enable "end-to-end" secure wallets and payments to fund escrow transactions or other complex transactions in a way that is backwards-compatible for old clients and miners.

## Specification

OP_EVAL will re-define the existing OP_NOP1 opcode, and will function as follows:

- When executed during transaction verification, pops the item from the top of the stack, deserializes it, and executes the resulting script.
- If there is no item on the top of the stack or the item is not a valid script then transaction validation fails.
- If there are any OP_CODESEPARATORs in the deserialized script then transaction validation fails.
- If there are any OP_EVALs in the deserialized script they are also executed, but recursion is limited to a depth of 2.
- Transaction verification must fail if interpreting OP_EVAL as a no-op would cause the verification to fail.

A new standard transaction type (scriptPubKey) that is relayed by clients and included in mined blocks is also defined:

```
DUP HASH160 {20-byte-hash-value} EQUALVERIFY OP_EVAL
```

Which is redeemed by a standard scriptSig:

```
...signatures... {serialized script}
```

Transactions that redeem standard OP_EVAL scriptPubKeys are only considered standard if the *serialized script* is, itself, one of the standard transaction types.

## Rationale

OP_EVAL allows the receiver of bitcoins to specify how they can be spent when they are spent, instead of requiring the sender of the bitcoins to know the details of how the bitcoins may be redeemed. The sender only needs to know the hash of the *serialized script*, and one new type of bitcoin address can be used to fund arbitrarily complex transactions.

If *serialized script* is a large or complicated multi-signature script, then the burden of paying for it (in increased transaction fees due to more signature operations or transaction size) is shifted from the sender to the receiver.

The main objection to OP_EVAL is that it adds complexity, and complexity is the enemy of security. Also, evaluating data as code has a long record of being a source of security vulnerabilties.

That same argument can be applied to the existing Bitcoin 'scripting' system; scriptPubKeys are transmit as data across the network and are then interpreted by every bitcoin implementation. OP_EVAL just moves the data that will be interpreted. It is debatable whether or not the entire idea of putting a little interpreted expression evaluation language at the core of Bitcoin was brilliant or stupid, but the existence of OP_EVAL does not make the expression language less secure.

There is a 1-confirmation attack on old clients that interepret OP_EVAL as a no-op, but it is expensive and difficult in practice. The attack is:

1. Attacker creates an OP_EVAL transaction that is valid as seen by old clients, but invalid for new clients.
2. Attacker also creates a standard transaction that spends the OP_EVAL transaction, and pays the victim.
3. Attacker manages to mine a block that contains both transactions. If the victim accepts the 1-confirmation payment, then the attacker wins because both transactions will be invalidated when the rest of the network overwrites the attacker's invalid block.

The attack is expensive because it requires the attacker create a block that they know will be invalidated. It is difficult because bitcoin businesses should not accept 1-confirmation transactions for higher-value transactions.

## Backwards Compatibility

Surprisingly, because OP_EVAL redefines the OP_NOP1 opcode, standard OP_EVAL transactions will validate with old clients and miners. They will check only that the *serialized script* hashes to the correct value; the OP_EVAL will be interpreted as a no-op, and as long as the hash is correct the transaction will be considered valid (no signature checking will be done by old clients and miners).

Old clients will ignore OP_EVAL transactions and transactions that depend on them until they are put into a block by either an old miner that includes non-standard transactions in its blocks or by a new miner.

Avoiding a block-chain split by malicious OP_EVAL transactions requires careful handling of two cases:

1. An OP_EVAL transaction that is invalid for new clients/miners but valid for old clients/miners.
2. An OP_EVAL transaction that is valid for new clients/miners but invalid for old clients/miners.

For case (1), new clients and miners will be coded to interpret OP_EVAL as a no-op until February 1, 2012. Before then, miners will be asked to put the string "OP_EVAL" in blocks that they produce so that hashing power that supports the new opcode can be gauged. If less than 50% of miners accept the change as of January 15, 2012 the rollout will be postponed until more than 50% of hashing power supports OP_EVAL (the rollout will be rejected if it becomes clear that a majority of hashing power will not be achieved).

For case (2), new clients and miners will be written to make sure that transactions involving OP_EVAL are valid if OP_EVAL is interpreted as a no-op. Example of a transaction that must fail for both old and new miners/clients:

```
 scriptSig:  {serialized OP_11}
scriptPubKey:  OP_EVAL OP_11 OP_EQUAL
```

## Reference Implementation

https://github.com/gavinandresen/bitcoin-git/tree/77f21f1583deb89bf3fffe80fe9b181fedb1dd60

## See Also

https://bitcointalk.org/index.php?topic=46538

"Bitcoin Address 01" BIP

M-of-N Multisignature Transactions BIP 11