[Bitcoin-development] The difficulty of writing consensus critical code: the SIGHASH_SINGLE bug

Peter Todd pete at petertodd.org
Thu Nov 6 21:32:15 UTC 2014

Recently wrote the following for a friend and thought others might learn
from it.

> *Nope, never heard that term. By "bug-for-bug" compatibility, do you mean*
> *that, for each version which has a bug, each bug must behave in the *same**
> *buggy way?*

Exactly. tl;dr: if you accept a block as valid due to a bug that others reject,
you're forked and the world ends.

Long answer... well you reminded me I've never actually written up a good
example for others, and a few people have asked me for one. A great example of
this is the SIGHASH_SINGLE bug in the SignatureHash() function:

```
    uint256 SignatureHash(CScript scriptCode, const CTransaction& txTo, unsigned int nIn, int nHashType)
    {

<snip>

        else if ((nHashType & 0x1f) == SIGHASH_SINGLE)
        {
            // Only lock-in the txout payee at same index as txin
            unsigned int nOut = nIn;
            if (nOut >= txTmp.vout.size())
            {
                printf("ERROR: SignatureHash() : nOut=%d out of range\n", nOut);
                return 1;
            }
<snip>

        }

<snip>

        // Serialize and hash
        CHashWriter ss(SER_GETHASH, 0);
        ss << txTmp << nHashType;
        return ss.GetHash();
    }
```

So that error condition results in SignatureHash() returning 1 rather than the
actual hash. But the consensus-critical code that implements the CHECKSIG
operators doesn't check for that condition! Thus as long as you use the
SIGHASH_SINGLE hashtype and the txin index is >= the number of txouts any valid
signature for the hash of the number 1 is considered valid!

When I found this bug¹ I used it to fork bitcoin-ruby, among others.
(I'm not the first; I found it independently after Matt Corallo) Those
alt-implementations handled this edge-case as an exception, which in
turn caused the script to fail. Thus they'd reject blocks containing
transactions using such scripts, and be forked off the network.

You can also use this bug for something even more subtle. So the
CHECKSIG* opcode evaluation does this:

```
    // Drop the signature, since there's no way for a signature to sign itself
    scriptCode.FindAndDelete(CScript(vchSig));
```

and CHECKMULTISIG* opcode:

```
    // Drop the signatures, since there's no way for a signature to sign itself
    for (int k = 0; k < nSigsCount; k++)
    {
        valtype& vchSig = stacktop(-isig-k);
        scriptCode.FindAndDelete(CScript(vchSig));
    }
```

We used to think that code could never be triggered by a scriptPubKey or
redeemScript, basically because there was no way to get a signature into a
transaction in the right place without the signature depending on the txid of
the transaction it was to be included in. (long story) But SIGHASH_SINGLE makes
that a non-issue, as you can now calculate the signature that signs '1' ahead
of time! In a CHECKMULTISIG that signature is valid, so is included in the list

of signatures being dropped, and thus the other signatures must take that
removal into account or they're invalid. Again, you've got a fork.

However this isn't the end of it! So the way FindAndDelete() works is as
follows:

```
    int CScript::FindAndDelete(const CScript& b)
    {
        int nFound = 0;
        if (b.empty())
            return nFound;
        iterator pc = begin();
        opcodetype opcode;
        do
        {
            while (end() - pc >= (long)b.size() && memcmp(&pc[0], &b[0], b.size()) == 0)
            {
                pc = erase(pc, pc + b.size());
                ++nFound;
            }
        }
        while (GetOp(pc, opcode));
        return nFound;
    }
```

So that's pretty ugly, but basically what's happening is the loop iterates
though all the opcodes in the script. Every opcode is compared at the *byte*
level to the bytes in the argument. If they match those bytes are removed from
the script and iteration continues. The resulting script, with chunks sliced
out of it at the byte level, is what gets hashed as part of the signature
checking algorithm.

As FindAndDelete() is always called with CScript(vchSig) the signature
being found and deleted is reserialized. Serialization of bytes isn't
unique; there are multiple valid encodings for PUSHDATA operations. The
way CScript() is called the most compact encoding is used, however this
means that if the script being hashed used a different encoding those
bytes are *not* removed and thus the signature is different.

Again, if you don't get every last one of those details exactly right, you'll
get forked.

...and I'm still not done! So when you call CScript(vchSig) the relevant code
is the following:

```
    class CScript : public std::vector<unsigned char>
    {

<snip>

        explicit CScript(const CScriptNum& b) { operator<<(b); }

<snip>

        CScript& operator<<(const std::vector<unsigned char>& b)
        {
            if (b.size() < OP_PUSHDATA1)
            {
                insert(end(), (unsigned char)b.size());
            }
            else if (b.size() <= 0xff)
            {
                insert(end(), OP_PUSHDATA1);
                insert(end(), (unsigned char)b.size());
            }
            else if (b.size() <= 0xffff)
            {
                insert(end(), OP_PUSHDATA2);
                unsigned short nSize = b.size();
                insert(end(), (unsigned char*)&nSize, (unsigned char*)&nSize + sizeof(nSize));
            }
            else
            {
                insert(end(), OP_PUSHDATA4);
                unsigned int nSize = b.size();
                insert(end(), (unsigned char*)&nSize, (unsigned char*)&nSize + sizeof(nSize));
            }
            insert(end(), b.begin(), b.end());
            return *this;
        }

<snip, rest of class definition>

    }
```

Recently as part of BIP62 we added the concept of a 'minimal' PUSHDATA
operation. Using the minimum-sized PUSHDATA opcode is obvious; not so obvious
is that there are few "push number to stack" opcodes that push the numbers 0
through 16 and -1 to the stack, bignum encoded. If you are pushing data that
happens to match the latter, you're supposed to use those OP_1...OP_16 and
OP_1NEGATE opcodes rather than a PUSHDATA.

```
This means that calling CScript(b'\x81') will result in a non-standard
script. I know an unmerged pull-req² related to sipa's BIP62 work has
code in the CScript() class to automatically do that conversion; had
that code shipped we'd have a potential forking bug between new and old
versions of Bitcoin as the exact encoding of CScript() is consensus
critical by virtue of being called by the FindAndDelete() code!

Even had we made that mistake, I'm not sure how to actually exploit it...
FindAndDelete() is only ever called in a consensus-critical way with valid
signatures; the byte arrays 01, 02, ..., 81 are all totally invalid signatures.

The best I could think of would be to exploit the script verification
flag SCRIPT_VERIFY_STRICTENC by using the little-known hybrid-pubkey
encoding³, which I spent the past two hours looking at. However it isn't
even soft-fork safe in the current implementation!  All I could find was
a new DoS attack⁴, and it's not exploitable in an actual release due to
the pre-v0.10 IsStandard() rules. :(


[¹]: https://bitcointalk.org/index.php?topic=260595.0
[²]: https://github.com/bitcoin/bitcoin/pull/5091
[³]:
https://github.com/bitcoin/bitcoin/blob/cd9114e5136ecc1f60baa43fffeeb632782f2353/src/test/data/script_valid.json#L739

[⁴]: http://www.mail-archive.com/bitcoin-development@lists.sourceforge.net/msg06458.html

--
'peter'[:-1]@petertodd.org
0000000000000000019121d8632bcba14de98125e8a9affc7d07c760706ba3879
-------------- next part --------------
A non-text attachment was scrubbed...
Name: signature.asc
Type: application/pgp-signature
Size: 650 bytes
Desc: Digital signature
URL: <http://lists.linuxfoundation.org/pipermail/bitcoin-dev/attachments/20141106/8ea2ca39/attachment.sig>
```

- Previous message: [Bitcoin-development] Nakapay - Proposal for payment method using client generated paycodes and federated paycode servers
- Next message: [Bitcoin-development] The difficulty of writing consensus critical code: the SIGHASH_SINGLE bug
- **Messages sorted by:** [date] [thread] [subject] [author]

More information about the bitcoin-dev mailing list