Executive Audit

This is a guide to help MKR Holders verify executive spells in the Maker Protocol before voting for them.

This document contains four main sections:

1. Finding the Contract Code
2.
    - deals with locating the code for the executive in question.
3. Validating Executive Code and Copy
4.
    - deals with ensuring that the spell and associated copy is as expected.
5. Anatomy of a Spell Contract
6.
    - shows examples of the main sections you should expect to see in an executive spell.
7. A Non-Exhaustive Checklist
8.
    - includes a list of the major things to double check and how to do so.
9.

The key approach when reviewing an executive spell is that if any part of an executive spell seems suspicious or confusing in any way it is always better to double check before voting. Even if you are mistaken in your concerns, this informs us to add more detail to this document to prevent someone becoming confused in the same way in the future.

Finding the Contract Code

The link to the spell on etherscan can be found on the voting portal in the 'Proposal details' pane. Look for theSpell Address property:

?

Clicking on theSpell Address will open the spell in Etherscan, where the viewer can click on the Contract tab to view the code:

?

The contract code should be visible. If it is only possible to see bytecode (meaning that the contract is unverified) then the executive should be viewed as potentially malicious. In this case, cry foul in chat and encourage people NOT to vote for it until the verified contract code is made available.

Beyond looking in theGovernance Portal , current and past executive spell copy can also be found onGithub .

Validating Spell Code

Validate that the on-chain Etherscan code matches the correspondingarchive code or the mostrecent executive in Github. If it is not possible to locate the most recent spell, it may be because this has not been merged to the master branch before it goes live, in which case changes can be found in the pendingpull requests .

To do this code comparison, use a program calledDiff Checker to compare the Etherscan spell code against the relevant Github code. Clickingfind difference will illustrate any differences between the two.

Due to name collisions when combining source files into the combined spell code, some interface names may differ between the github version and the on-chain version. In this case, you may see_1 or_2 appended to certain interface names in the on-chain Etherscan code, this is not a cause for alarm.

Note, the Etherscan code will also illustrate interfaces to the DssExecLib, represented by the additional green section at the top of the file, which includes the following for awareness:

```

Copy The following library code acts as an interface to the actual DssExecLib library, which can be found in its own deployed contract. Only trust the actual library's implementation.

```

This means that it is the DssExecLiblibrary itself that should be reviewed, and is something that will be covered in the following spell anatomy section. Otherwise, everything below this green text should be identical to the spell contents as presented in Github.

?

In the above example, the comparison is between thisGithub file against the correspondingEtherscan contract code .

Validating Governance Copy

To ensure that the various actors (including governance facilitators, and smart contract developers) are aligned, the body of the spell includes a hash of the github copy that appears on the governance portal. This hash is included in the spell to confirm that the smart contract developers building the spell have referenced the correct copy being voted on. The hash appears in the spell in this manner:

```

Copy "2021-09-24 MakerDAO Executive Spell | Hash: 0x655e71cb00b63a11c14db615d43d3e59202ff9d2b2bc6a6a03de42e258bd1be8";

```

To validate that the governance portal copy on vote.makerdao.com which is reflected onGithub has the same hash as is stored in the executive spell, the user can copy and paste the body of the gituhub text into this onlineGithub tool to generate and match the hash.

Alternatively, the user can generate this hash run the following seth command:

```

Copy seth keccak -- "(wget https://raw.githubusercontent.com/makerdao/community/f33870a7938c1842e8467226f8007a2d47f9ddeb/governance/votes/Executive%20vote%20-%20October%208%2C%202021.md -q -O - 2>/dev/null)"

```

If the hash does not match, it would be prudent to question the spell contents and the copy that was referenced or changed.

Anatomy of an Executive

An executive spell is broken into a number of different parts, including; spell libraries, spell actions, and constructors, which will be reviewed below.

Spell Library

MakerDAO's executive spells have evolved over time towards using libraries in an effort to automate the process of making repetetive updates and changes to protocol parameters. Libraries reduce the chance of errors by reusing or building upon prior tested and verified code.

In auditing the executive it is first necessary to check that the spell uses the dss-exec-lib library and secondly, that the library is the official and legitimate library. This can be confirmed by looking at the spell copy in Etherscan.

The user can confirm that the executive is using thedss-exec-lib , by identifying the entry point for DssSpell is DssExec at the bottom of the contract code:

```

Copy contract DssSpell is DssExec

```

It is then necessary to validate that the correct library is linked to and therefore importing the right code. By remaining on the etherscan contract tab, one can scroll to the bottom of the page and view the address of the DssExecLib:

?

To verify that DssExecLib is not malicious, the user can take the code block and compare it against a known repo, such asmakerdao/dss-exec-lib to validate any changes that may be present. Using Diff Checker is again useful for this purpose.

It is also important to compare the address of the ExecLib from Etherscan against the MakerdaoMakefile and ensure that it matches the DssExecLib address:

?

Spell Constructor

Now that the user is satisfied that the library is valid, it is possible to return to the spell and review the last line which reveals the DssExec constructor which determines the schedule and cast constraints:

```

Copy { constructor() DssExec(block.timestamp + 30 days, address(new DssSpellAction())) public {} }

```

This line calls the constructor code inside the DssExec Library and passes it an expiration timestamp, so whenever this spell is created, it takes the current block timestamp and adds a number of days, in this case 30 days. This should align with Governance process. It is also necessary to ensure that this passes the above DssSpellAction code block which instantiates the spell.

It is worth noting that although the constructor determines schedule and cast constraints, these variables have been abstracted away into theExecLib , and no longer appear in the spell.

Afterschedule has been called and after the delay is up, anyone can callcast which will execute the code in the SpellAction'sDssSpellActions function, below.

Spell Actions

Now that there is confidence in the framework around the spell and the external library, it is possible to focus on Spell Actions, which is the bulk of what is getting scheduled and cast for execution.

Users reviewing past spells may recall the use of theexecute function. This is no longer used as Spell Actions are used to point to the ExecLib to make protocol changes. Again this highlights the importance of the library and a useful approach for reducing the size of spells.

As a general rule, it is important to check everything in the section before or after the Spell Actions is aconstant , as shown here for example:

```

Copy contract DssSpellAction is DssAction {

// Provides a descriptive tag for bot consumption // This should be modified weekly to provide a summary of the actions // Hash: seth keccak -- "(wget https://raw.githubusercontent.com/makerdao/community/3d3670a98ca02e74c001bdf083765ab4eb4bc2cd/governance/votes/Executive%20vote%20-%20September%2024%2C%202021.md -q -O - 2>/dev/null)" string public constant override description = "2021-09-24 MakerDAO Executive Spell | Hash: 0x655e71cb00b63a11c14db615d43d3e59202ff9d2b2bc6a6a03de42e258bd1be8";
```

// L2 Test Spells address constant OPTIMISM_L1_GOVERNANCE_RELAY = 0x09B354CDA89203BB7B3131CC728dFa06ab09Ae2F; address constant OPTIMISM_L2_SPELL = 0x71d75C3D100D14d4db0cE7a83d0De48ecEC32D19; address constant ARBITRUM_L1_GOVERNANCE_RELAY = 0x9ba25c289e351779E0D481Ba37489317c34A899d; address constant ARBITRUM_L2_SPELL = 0xAeFc25750d8C2bd331293076E2DC5d5ad414b4a2;

// Math uint256 constant MILLION = 10 ** 6;

function actions() public override {

```

If any of these values are not constants, they could present malicious code in the form of memory mutations impacting the function of the pause proxy, and the community should be alerted. This risk is further covered below in the Constant Variables section.

As mentioned in the library section above, standard actions will typically use DssExecLib calls to perform an action in a safe way. For non-standard actions, these will fall back to direct system calls. Some examples include:

```

Copy function actions() public override {

// Set ilk/global DC DssExecLib.increaseIlkDebtCeiling(collateral.ilk, collateral.CEIL, true);

// Set stability fee DssExecLib.setIlkStabilityFee(collateral.ilk, collateral.RATE, false);

// Set values directly Fileable(col.dog).file(col.ilk, "hole", col.hole); Fileable(col.dog).file(col.ilk, "chop", col.chop); Fileable(col.clipper).file("buf", col.buf);

}

```

Office Hours

The office hours function determines whether or not the spell, once voted in, is limited to being scheduled between 10:00 - 16:00 EST weekdays. Setting the value to true/false specifically enumerates whether it is applied or not.

If this function is missing from the Spell Action, the code will fall back to the default office hours state there, which is true (office hours on)

```

Copy function officeHours() public virtual returns (bool) { return true;

```

Such a value istrue when new changes may impact integrators or auction keepers, and is otherwisefalse when office hours are not applied - meaning the spell can be cast at anypoint once successfully voted in. The code determining these rules has also beenabstracted away into the ExecLib.

Non-Exhaustive Checklist

Verify that the contract code is visible

Using the instructions above, find the contract code on Etherscan and ensure that it is visible and verified. Similarly ensure that the ExecLib is correctly referenced. Verify that these match what the smart contract developers have published/deployed.

Verify that the frontend copy matches the copy used to build the spell

Create a hash of the github/frontend copy and reference this against the hash in the spell to confirm that the two align.

Verify the constructor is as expected

Ensure that the spell's expiry matches previous spells, includes the DssActions code block and that schedule and cast functions exist in DssExecLib. If the Constructor does not match previous functions, there should be an explanation as to why.

Review the changelog

Ensure that DssExecLib calls out to the changelog with the correct name descriptors. (These used to be spell addresses, but have since been replaced with name descriptors for error reduction.) It is therefore good practice to review the changelog to ensure that the correct name descriptor has been referenced as part of the on-chain action.

Review oracle addresses

Occasionally, there are spells involving Oracles - generally adding addresses to the whitelist, and occasionally adding a new oracle. These addresses must also be carefully verified. To do so it is necessary to go to theChangelog and look for e.g.PIP_ETH . Taking the contract to Etherscan and opening the contractread tab will display thesrc (listed as number 7 currently). The contract address listed there can be verified to match theMedianETHUSDcontract that is in the spell.

Review rate changes

Rates are defined as per-second accumulation values. These values can be validated against the commented rate by using the bc command in a bash shell. Using theNEW_FEE variable in the example contract the following is visible:

bc -l <<< 'scale=27; e( l(1.095)/(60 * 60 * 24 * 365) )'

This produces 1.000000002877801985002875644. Removing the decimal place will allow you to see that this matches the definition ofNEW_FEE .
```

Validating all rate adjustments can be done the same way. For more information on the rates module, please visit thedeveloper guide . For easy reference, common pre-computed rates can also be viewed at the followingipfs link .

Review DssExecLib to ensure Drip is called before rates are changed

Immediately prior to making rate changes,drip must be called on the respective contracts. For example, if a DSR rate change is being made,drip is called on thepot or if the stability fee is being changed on a collateral type, it is necessary to calldrip("ILK") on thejug . Despitedrip being abstracted to the DssExecLib, it can still be confirmed in the library as part of the prior libary checks.

To allow for situations wheredrip is unwanted, the execlib functions included a boolean parameterdoDrip . Users should ensure eitherdoDrip istrue or there is a good reason why not. For example, see the codehere ;

```

Copy function setIlkStabilityFee(bytes32 _ilk, uint256 _rate, bool _doDrip) public {

```

All SpellAction contract variables must be declared 'constant'

SpellActions must never have anything in contract memory. Therefore, all contract variables must be declared as constant. This is because at execution time, the contract's variables will be that of the DSPauseProxy. If there are variables in this section that are anything other than constant then it is a significant bug and must not be approved.

Page last reviewed: - Next review due: -

Previous Executive Vote Considerations Next Executive Transactions Last updated9 months ago On this page *Finding the Contract Code * Validating Spell Code * Validating Governance Copy * Anatomy of an Executive * Spell Library * Spell Constructor * Spell Actions * Office Hours * Non-Exhaustive Checklist * Verify that the contract code is visible * Verify that the frontend copy matches the copy used to build the spell* Verify the constructor is as expected * Review the changelog * Review oracle addresses * Review rate changes * Review DssExecLib to ensure Drip is called before rates are changed * All SpellAction contract variables must be declared 'constant'

Was this helpful? Edit on GitHub