

Forum:[Hack Forums](#)

Thread: [Seeking help to bypass MD5 Checksum / Hash validation in Lua 5.0.3 \(MTA\)](#)

Drayzen705 - February 9, 2026, 8:05 am

Hi everyone,

I'm currently analyzing a game server (MTA) that uses Lua 5.0.3 for its security resources. I've encountered a challenge with the server's file integrity check.

The Problem: Whenever I attempt to modify a client-side script, the server detects the change and throws a "Wanted MD5" mismatch error, preventing me from joining. It seems the server cross-checks the local file hash with a pre-defined value before execution.

Key Observations:

The server has duplicate security files (e.g., two files named trigger_security.lua in the same directory), possibly creating a race condition.

Some resource files are masked as images (e.g., 39.png) but are recognized as scripts by the engine.

Static file modification is blocked by the MD5 enforcement.

My Question: What is the best approach to bypass this hash validation? Should I look into Memory Hooking to intercept the checksum function, or is there a way to spoof the MD5 response during the handshake?

If anyone has experience with Resource Injectors or Lua bypasses for this specific engine version, I would appreciate any guidance or tools.

Thanks in advance!

DramaKing - February 9, 2026, 9:54 am

First observation is that identically named files in the same directory is impossible on every OS. Second, you probably need a hash collision. Unless you can alter the response to the server so that the file integrity check passes.

Drayzen705 - February 16, 2026, 11:21 am

(Feb 9, 2026 10:54 AM)DramaKing Wrote: [\[CITE\]](#) First observation is that identically named files in the same directory is impossible on every OS. Second, you probably need a hash collision. Unless you can alter the response to the server so that the file integrity check passes.

I've completed a technical audit of the MTA server resources and found some interesting vectors, but the security implementation is quite robust. Here is what I've identified and executed so far:

Collision/Race Condition: I noticed two files named trigger_security.lua in the grp_vehicleengine folder. I suspect a hidden character or null byte manipulation, which could potentially trigger a Race Condition during resource initialization.

Resource Masking: Found obfuscated code hidden as 39.png. The engine treats it as a script,

confirming a Steganography/Masking bypass.

Integrity Barrier: I hit a wall with the MD5 Integrity Check. Any attempt to modify client-side scripts results in a mismatch error (specifically requesting hash: 6E97D4...). The server-side validation is strictly enforced before the handshake completes.

Given these findings and your expertise in CEH and DFE, what would you suggest as the next logical step? Since the MD5 check is the primary blocker, should I look into:

Memory Injection: Hooking the MTA process to manipulate the script after it passes the integrity check?

Hash Collision: Attempting to generate a modified script that matches the target MD5 (though computationally expensive)?

Handshake Interception: Using a proxy to spoof the hash response to the server?

I'd appreciate your detailed insight on how to move past this integrity enforcement

Nomad.Null - February 17, 2026, 12:11 pm

(Feb 16, 2026 12:21 PM)Drayzen705 Wrote: [\[CITE\]](#) I've completed a technical audit of the MTA server resources and found some interesting vectors, but the security implementation is quite robust. Here is what I've identified and executed so far:

Collision/Race Condition: I noticed two files named trigger_security.lua in the grp_vehicleengine folder. I suspect a hidden character or null byte manipulation, which could potentially trigger a Race Condition during resource initialization.

Resource Masking: Found obfuscated code hidden as 39.png. The engine treats it as a script, confirming a Steganography/Masking bypass.

Integrity Barrier: I hit a wall with the MD5 Integrity Check. Any attempt to modify client-side scripts results in a mismatch error (specifically requesting hash: 6E97D4...). The server-side validation is strictly enforced before the handshake completes.

Given these findings and your expertise in CEH and DFE, what would you suggest as the next logical step? Since the MD5 check is the primary blocker, should I look into:

Memory Injection: Hooking the MTA process to manipulate the script after it passes the integrity check?

Hash Collision: Attempting to generate a modified script that matches the target MD5 (though computationally expensive)?

Handshake Interception: Using a proxy to spoof the hash response to the server?

I'd appreciate your detailed insight on how to move past this integrity enforcement

I have to agree with @DramaKing. You'll need to find a hash collision by adding a comment string at the bottom of the script and continuously modify it and run an MD5 checksum until you find a collision. It's definitely expensive, but feasible with MD5 because it's cryptographically a broken mess. Birthday attacks on MD5 are actually very doable. If you want, I can code up a rust binary that modifies a lua file with a comment string at the bottom and test it against your benign script version until it finds a collision. It can be done entirely in memory without continuously re-writing the file to

spare some of the computational energy.

There is always a chance that the full md5 128-bit hash doesn't get checked, but rather just the first half or so, which dramatically increases your chance of finding a collision.

Edit:

I'm willing to do write the code if you are willing to pay for the work. Depends on how important this is to you.

I was bored and decided to write the code for fun.

If you want it, I'll sell it to you. It will work eventually but relies on CPU to do the hashing, so it won't be nearly as efficient. I'm working on adding GPU-cracking support into it because why the hell not? I was able to write this in about 20 minutes so far, so adding CUDA support shouldn't take long. Let me know if you want it.

After determining that even with a really powerful GPU, you're looking at about 9 years of hashing to find a collision at 2^{128} bits. Not as feasible as I had previously considered.

As a side experiment, I did look into the possibility of using a proxy to intercept the server-side file validation. It all depends on whether the file integrity validation check happens client-side and it just sends the response to the server. I don't know what it is you're modifying so without looking at it personally, I obviously can't determine that, but in theory it may be possible. Chances are pretty damn good, the server isn't downloading the file from the client-side and running a hash checksum on it to verify you're actually loading the correct files. Most likely, it hooks a native library, runs the checksum on the files, and sends those checksums to the server. If the validation happens in a lua script, you could do something like create a hashmap of the original files you've modified, overwrite the script that performs the validation, and have it replace any result you give with the original file's hash before sending it server-side.

If a compiled binary is what is doing the validation, you're looking at some serious reverse-engineering to modify the binary which is gonna be a lot harder and probably more trouble than it's worth.

In that case, you could proxy all your traffic to the server that validates the files through something like burpsuite and figure out how to intercept, tamper with, and inject the valid hashes en route before it reaches the server.

Any way you slice it, this is a complex problem. You let me know how you'd like to proceed.

Proof of Concept for that birthday attack program. I've written it in Rust. Highly efficient and optimized for single-target MD5 collisions. With my RTX 4080m it can crush the first 32-bits of an MD5 hash in under 30 seconds. 48-bits takes about 16-20 hours. The full 128 bits is gonna take about $3e9^{10}$ years though ?

Thanks for the fun little code project idea.