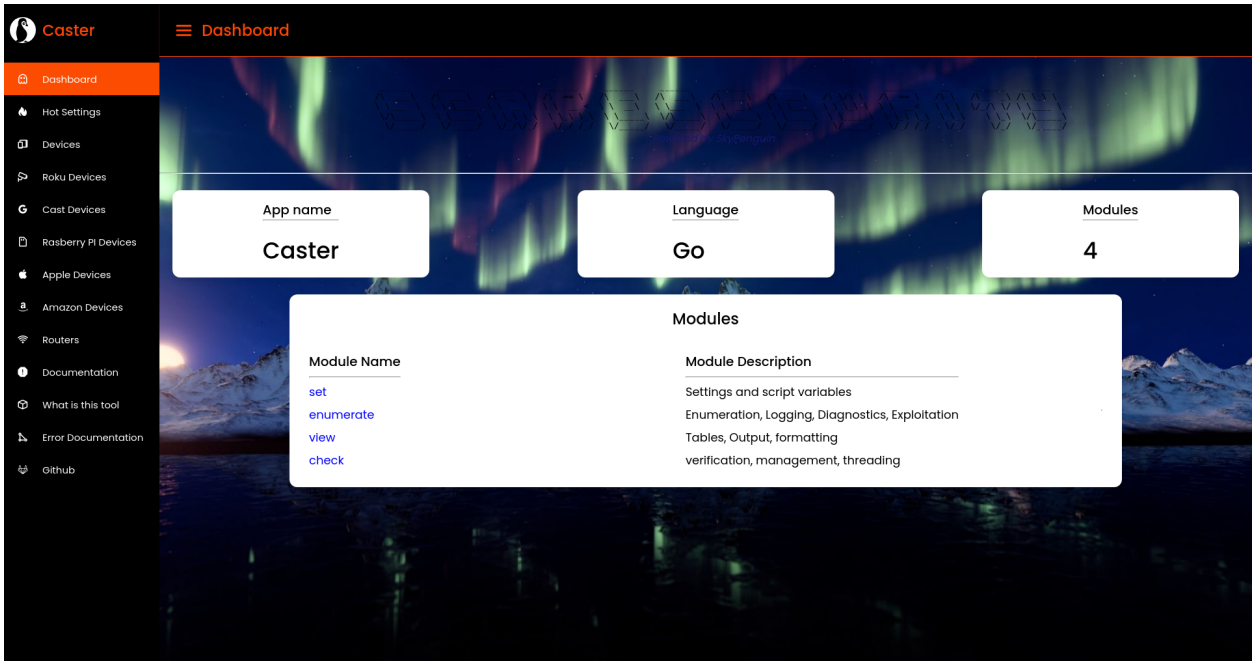
Caster - The IoT manipulation framework | A test of SkyLine's capabilities



About | Technical Information (Document)

This document is much more different than all of the other SkyLine technical documents. This document is here for a few reasons so let me break it down for you. SkyLine plans to be used for cybersecurity-related tasks and mathematical relative tasks, in order to test that, it is best to take Totally_Not_A_Haxxers (The founder of SkyLine and SkyPenguin) most popular and largest frameworks and rewrite them using SkyLine. This will test the capabilities in SkyLine and address problems that can only be seen by a user of the language rather than the developer. This document will outline why SkyLine will be used, where Golang will come into play, the technical specs of caster, why caster exists, the problem where Golang becomes required for SkyLine to run properly, and also outline prime issues in SkyLine before re-doing the entirety of SkyLine to include SkyVM - SkyBC and an entire revamp of the engine and its utilities. This document will also help prove where SkyLine can come into play, where it fails, and where its idea becomes the utmost acceptable.

> Note: As with most documents and as already said SKYLINE IS AN EXPERIMENT AND IS NOT A PRODUCTION READY OR PURPOSELY USED LANGUAGE. IT IS USED TO REMAKE CURRENT PROJECTS TO TEST THE ABILITY OF THE LANGUAGE AND TO TEST AND BREAK BARRIERS. IF YOU USE SKYLINE, USE THE MOST CURRENT PUBLIC OPEN-SOURCED VERSION NOT THE DEVELOPER VERSION. VERSIONS LIKE DEVELOPER VERSIONS ARE FLOODED WITH BUGS AND EXPERIMENTS THAT HAVE NOT BEEN FULLY TESTED OR WORKED WITH.

Caster | What is Caster (Module 1)

The caster framework was a project started by (Totally_Not_A_Haxxer) to abuse the API endpoints that were discovered in IoT devices by being able to find devices on the network and automate requests and payloads to endpoints that are not normally accessed in that same way per se. This framework's goal was to help security experts easily grab information from devices to better help their exploitation process and to also better help their security research towards IoT devices like routers, cars, radios, smart TVs, cameras, and hundreds of devices above that. Caster would work by starting a sub thread on your system and crafting ARP (Address Resolution Packets) and sending them to every single host on the network on which the scanner was set. The scanner would choose a random unused and most efficient device from a scan it does before running main functions. When these packets are sent a secondary thread is started to listen for ARP-RESP (Address Resolution Protocol Response Packets) which would then be individually parsed on a third thread. This thread responsible for parsing it would go through multiple verification stages before actually finishing everything. This means that the thread would not only check and verify layers but look for very specific protocols according to the current and most used ARP standard. It would then dissect the major components such as the MAC of the device, the source address, the destination address, the OUI of the device, and other information like timestamps that may be useful during parsing. When this stage hits it would check the OUI of the device and parses for information, if it matches a supported manufacturer name such as Amazon, Google, Microsoft, Samsung, Roku, and other various supported manufacturers, it would then be added to a list which would be kept separate from the base list of discovered hosts which may possibly be other IoT devices. This list separation would allow you to view all Amazon targets, google targets, and other various sets of information basically sub-parsing the list. Caster would also start a secondary background thread and process which would listen for SSDP packets coming from Amazon devices because it could then find UUIDs or session IDs through the SSDP packets which could then be used to access other system API endpoints. This framework would allow you to abuse the API endpoints of different devices by probing the endpoints for information, changing and configuring information or settings, querying for device and Bluetooth information, sniffing specific packets and saving them for better parsing, manipulating responses, and so on from there. Caster also had a web interface that would allow you to view all of your hosts and also would show documentation for Caster and how Caster worked as a framework. Caster required a ton of heavy functionality such as multi-threading, multi-processing, socketry, networking, access to protocols and implementations, dissection of binary files, dissection of API’s and even had modules to parse AppleTV response files from servers or locate and deobfuscate/decode BPLIST files. This helped a ton when automating all of the requests but it had one major issue, it was a console- not really a tool that could just run in the background. In later versions of its framework and its design, it is supposed to run in the background on your system, output its information and capture as well as let the user know what to and to not do. The major issue with Caster as well was its art, color, and systems and the organization was generally horrible so the developer never went back to the project and abandoned it for a little bit.

SkyLine | How does it come into play when re-writing Caster? (Module 2)

Caster in itself as explained above was a very dependent tool, it required access to multiple libraries and automated a bunch of system calls which were just useless bloat. With the development and updates to SkyLine we believe that with proper care and design and attention to detail (this document) Caster can be fully rewritten with golang as well. This means that the language would use Shared Object (so) files to be plugged directly into the SkyLine project and that the engine could also work with Golang Shared Object Files as plugins to the project. SkyLine already has an okay-sized base but the idea is being slowly developed over time. In order to prove that its libraries and its build is designed directly for cybersecurity-related tasks. With this idea in mind, SkyLine will be pushed to its absolute max to develop one of the bigger frameworks that were written and released. When Caster is pushed out in the SkyLine version the original version with statistics and load time will also be compared. You might be asking, how can you compare something like Caster which was written in Golang to an interpreted not even byte code compiled language like SkyLine? The purpose is not to compare the functionality of the Go programming language nor is it to compare the speed of the two languages but rather compare the results of before and after. These tests will include the following questions which should be answered by the end of rewriting the Caster framework.

- What issues did SkyLine have during development?
- What did SkyLine do better?
- Did SkyLine provide a better backend for this task?
- Was it faster to write and develop than before?
- What tasks were encountered?
- What changes did you have to make to SkyLine's backend to complete development?
- What changes did you make to the engine to support Caster?
- What all in all was the most important role?
- Did any crashes happen? If so where were they?
- How fast did SkyLine work with the backend and the front end?
- Were the unregistering libraries helpful?
- Was the module system easy to work around?
- What should the language require that would make it better for this task?
- What would make the language more cyber security related?
- What plus do you think plugins bring to SkyLine?
- Do you think the plugins are a great way to interact with projects?
- Was the runtime viable or would using SkyVM be better?
- In the future, does this help prove SkyLine's end goal?

Answering these questions will contribute A TON to SkyLine. Along with the Caster re-development phase will also bring a ton of contribution to the SkyLine development side of things. This is because these questions are mainly about how SkyLine performed and the analysis of tasks is also one of the more important things. Given that SkyLine is interpreted already shows that the language will be easier in terms of development times, but the major issue is why is SkyLine so good for cyber security. Caster again will be able to better prove this

idea of using SkyLine for cyber security over something like Python. Now then again, the end results may be much more different because SkyLine will have a backend developed specifically for these use cases which Caster itself implements. This is also why SkyLine in itself while having a much more developed and formulated idea now is going to be a pain in the ass to actually prove. In this day and age languages like Perl, Ruby, C, C++, Python, and Go are almost impossible to prove wrong because each of those languages has been around for such a long time and was developed by teams a year after their initial release or started out with hundreds of people contributing to the language and developers or sponsors ready to fund the project. This makes proving that language's wrong nearly impossible right now, but the point is not to prove the language wrong but again rather prove why SkyLine's backend and general tailor towards cybersecurity makes this language better for specific operations especially given that it can also run golang plugins right into the program. This plugin system also changes things up given the majority of caster will still be written in Go, that is at least on its networking side. Right now, SkyLine does not at all have any shape or form of a library that can allow you to manually create sockets or make connections to hosts or even yourself which means there is no packet parsing or capturing library. Because SkyLine will take quite a while before it even becomes production ready to do something like that, plugins will come into play when using GoPacket (A packet-capturing library used by Caster developed by Google for Golang). Because of this plugin system and the ease of tying everything together in a way, it will make the development on Caster much more complex but easier in itself. The end goal of this development is to also eliminate the use of third-party packages in the Caster framework which will better demo SkyLine's idea and end goal or range idea for the cyber security world. As previously discussed in **Document B (SkyLine - Technical Specification _ SkyPenguin Solutions.pdf)**, SkyLine's prime goal is to eliminate multiple things and those prime things are as follows.

- **The use of third-party software** or libraries for the smallest and even most advanced tasks by automating or providing a better framework for developers to easily automate those same tasks in just minutes of working with SkyLine.

- **The use of reckless bloat by third-party libraries.** By SkyLine providing that framework making it easier for developers to make their own third-party software with its standard library tailored to cybersecurity-related tasks, the endless bloat from third-party libraries relying on more third-party libraries which rely on their own third-party libraries will be inexistent.

The rewrite in Caster not only will be able to prove its point by removing 90% of the bloat slowing Caster down but will also be able to prove how sometimes useless third-party libraries can be and how SkyLine especially way further down the line can provide a much better interface/selection of frameworks to allow developers to easily make their own frameworks and third-party libraries which can easily work with Golang related projects.