Malware Development with Nim

A Case Study in NimPlant

Cas van Cooten

2023-03-23

[P]

00 | About

```
[cas@prelude-discord ~]$ whoami
```

- Offensive Security Enthusiast, Red Team Operator, and hobbyist Malware Developer
- Likes building malware in Nim
- Authored offensive tools such as Nimpackt, and more recently NimPlant
- Semi-pro shitposter on Twitter

Cas van Cooten

casvancooten.com

@chvancooten

chvancooten

/in/chvancooten

01 | Preface: Offensive Development

Build your own tools for fun and profit

Follow us

The Hacker News

Home | Cyber Attacks | Vulnerabilities | Offers | Contact

Researchers Spotted Malware Written in Programming Language

March 12, 2021 | Ravie Lakshmanan

Cybersecurity researchers have unwrapped an "interesting email campaign" u
actor that has taken to distributing a new malware written in Nim programmi

the executable (Figure 3):

.rdata:0000000...	0000005F	C	@iterators.nim(222, 11) `len(a) == L`
.rdata:0000000...	0000002C	C	@json.nim(293, 10) `father.kind == JA
.rdata:0000000...	00000029	C	@json.nim(367, 9) `obj.kind == JObje
.rdata:0000000...	00000013	C	monocypher_nim.nim
.rdata:0000000...	00000037	C	@monocypher_nim.nim(27, 12) `len(r
.rdata:0000000...	00000037	C	@monocypher_nim.nim(17, 12) `len(k
.rdata:0000000...	0000003B	C	@monocypher_nim.nim(33, 12) `len(r
.rdata:0000000...	00000025	C	@json.nim(485, 9) `not isNil(node)`
.rdata:0000000...	00000027	C	@utils.nim(12, 12) `len(a) == len(b)`
.rdata:0000000...	00000035	C	@crypto.nim(50, 12) `len(sig) == sizeof(Signature)`
.rdata:0000000...	0000000E	C	strformat.nim
.rdata:0000000...	00000022	C	@random.nim(568, 12) `seed != 0`
.rdata:0000000...	0000005C	C	@iterators.nim(204, 11) `len(a) == L` the length of the seq changed while iterating over it
.rdata:0000000...	0000002A	C	@json.nim(486, 9) `node.kind == JObject`
.rdata:0000000...	00000025	C	@json.nim(485, 9) `not isNil(node)`

nim

Figure 3: Example of Nim related strings

ZDNet

Home | Innovation | Security

Malware developers turn to 'exotic' programming languages to thwart researchers

They are focused on exploiting pain points in code analysis and reverse-engineering.

Written by **Charlie Osborne, Contributor** on **July 27, 2021**

Malware developers are increasingly turning to unusual or "exotic" programming languages to hamper analysis efforts, researchers say.

According to a new report published by BlackBerry's Research & Intelligence team on Monday, there has been a recent "escalation" in the use of Go (Golang), D (DLang), Nim, and Rust, which are being used more commonly to "try to evade detection by the security community, or address specific pain-points in their development process."

TLP:WHITE

FBI FLASH

EDERAL BUREAU OF INVESTIGATION • CYBER DIVISION

he following information is being provided by the FBI, with no guarantees or warranties, for potential se at the sole discretion of recipients to protect against cyber threats. This data is provided in order o help cyber security professionals and system administrators to guard against the persistent malicious ctions of cyber actors. This FLASH was coordinated with DHS/CISA.

his FLASH has been released **TLP:WHITE**

f you identify any suspicious activity within your enterprise or have related information, ur local FBI Cyber Squad immediately with respect to the procedures outlined in the Reporting Notice section of this message.

y related information to FBI Cyber Squads, you are assisting in sharing information that allows the FBI to track coordinate with private industry and the United States Government to prevent future intrusions and attacks.

/ALPHV Ransomware Indicators of Compromise

t of a series of FBI reports to disseminate known indicators of compromise (IOCs) and es and procedures (TTPs) associated with ransomware variants identified through FBI s of March 2022, BlackCat/ALPHV ransomware as a service (RaaS) had compromised at worldwide and is the first ransomware group to do so successfully using RUST, a more secure programming language that offers improved performance and reliable essing. BlackCat-affiliated threat actors typically request ransom payments of several Bitcoin and Monero but have accepted ransom payments below the initial ransom . Many of the developers and money launderers for BlackCat/ALPHV are linked to

g Malware Count Over Time

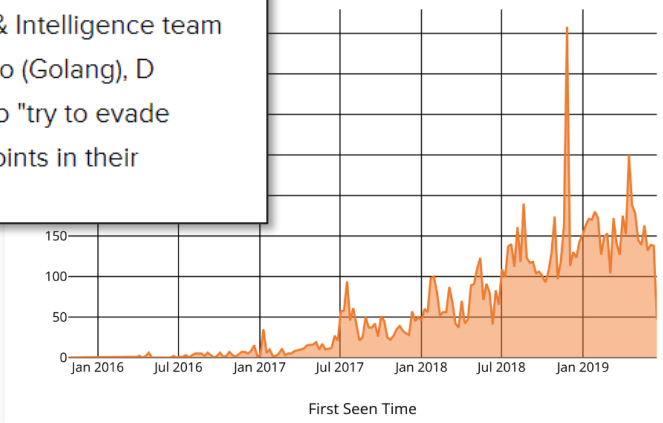


Figure 1. Timeline of Go Malware samples based on first seen dates.

02 | Nim

Nim for malware development

- Compiles directly to C, C++, Objective-C or Javascript
- Doesn’t rely on VM or runtime, yields small binaries
- Python-inspired syntax, rapid development and prototyping
  - Avoids you having to write C/C++ (goodbye vulns!)
- Has an extremely mature Foreign Function Interface (FFI)
- Super easy cross-compilation (using mingw)

github.com/byt3bl33d3r/OffensiveNim

02 | Nim

Nim to bypass defenses?

Virus scanner problems after installing Nim 1.4

Questions

wiltzutm Oct 2020

Hello, I'm having difficulties with my administered Windows 10 laptop's virus scanner (F-secure Client Security Premium) and the latest nim release 1.4. My previous nim version was 1.2.6.

So all was good with Nim 1.2.6, but after updating I began to get weird heuristics false alarms (HEUR/APC) from the scanner when compiling without the release flag: "nim c hello.nim". These false alarms won't pop up when I use the "nim c -d:release hello.nim" which I find odd.

I know this isn't your problem, **I was just wondering if there's someone who has some experience and under the hood understanding what MIGHT trigger some heuristics when compiling without the release flag?** I don't actually even know if it's even possible to speculate without seeing the scanner's code.

One problem at the moment is that I cannot even send a sample file to the virus scanner company because the scanner is so aggressively deleting the compiled example file... (of course I could do this with some other pc, but I don't have any windows pc at my possession only linux)

Is my only solution to break free from my workplace admins and start devving Nim with my non administered linux pc (I would prefer linux, but the laptop is so slow)? :D

File hello.nim contents:

```nim
proc sayHello() =
    echo "Hello World!"

when isMainModule:
    sayHello()
```

r/nim

Posts

Posted by u/al_earner 9 months ago

6

The virus issue

I'm trying to install Nim 1.6 (64 bit) on Windows 10. The download fails because Windows detects the Sabsik.FT.A!ml virus. I know there is some sort of known issue with this virus and Nim, but the file is 20 meg... that's a lot of space to hide a virus. Are we sure this is a false positive?

88% Upvoted

6 Comments Share Save Hide Report

02 | Nim

Nim to bypass defenses!

03 | Nim in Practice

Getting acquainted with the syntax

```
import base64
import httpclient

var client = newHttpClient()
let content = client.getContent("https://twitter.com/chvancooten")
let encoded = encode(content)

if encoded.len <= 64:
  echo encoded
else:
  echo encoded[0..31] & "..." & encoded[^32..^1]
```

rosettacode.org/wiki/Base64_encode_data#Nim

03 | Nim in Practice

WinAPI: P/Invoke

```
type
  HANDLE* = int
  HWND* = HANDLE
  UINT* = int32
  LPCSTR* = cstring

proc MessageBox*(hWnd: HWND, lpText: LPCSTR, lpCaption: LPCSTR, uType: UINT): int32
  {.discardable, stdcall, dynlib: "user32", importc: "MessageBoxA".}

Messagebox(0, "Work smart, not hard", "Hello Prelude!", 0)
```

Dynlib uses GetProcAddress + LoadLibrary,
D/Invoke by default!

github.com/byt3bl33d3r/OffensiveNim

03 | Nim in Practice

WinAPI: P/Invoke (for the lazy)

```
import winim/lean

Messagebox(0, "Work smart, not hard", "Hello Prelude!", 0)
```

03 | Nim in Practice

Using compile-time macros to obfuscate static strings

```
import strenc

echo "Hello world! Betcha can't find this string
in the compiled binary 👀"
```

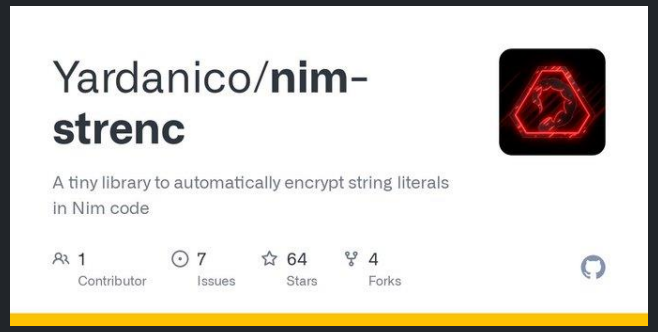
Decompile: NimMainModule - (MacroExample.exe)

```
15    local_28 = 0;
16    nimFrame(&local_48);
17    local_38 = 3;
18    local_30 = &DAT_00415408;
19    nimZeroMem(local_18,8);
20    local_18[0] = gkkaekgaEE__hGw89cXYiTNl4W5w8FOj49cw
21                              ((longlong *)&TM__hkPwWT6FKR9a59cAlSLCyE9ag_2,0x4f9d654f);
22    echoBinSafe((longlong)local_18,1);
23    popFrame();
24    return;
25  }
26
```

03 | Nim in Practice

Using compile-time macros to obfuscate static strings

```nim
import macros, hashes

type
  estring = distinct string

proc obfuscate(s: estring, key: int): string {.noinline.} =
  var k = key
  result = string(s)
  for i in 0 ..< result.len:
    for f in [0, 8, 16, 24]:
      result[i] = chr(uint8(result[i]) xor uint8((k shr f) and 0xFF))
    k = k +% 1

var encodedCounter {.compileTime.} = hash(CompileTime & CompileDate) and 0x7FFFFFFF

macro encrypt*{s}(s: string{lit}): untyped =
  var encodedStr = obfuscate(estring($s), encodedCounter)

  template genStuff(str, counter: untyped): untyped =
    {.noRewrite.}:
      obfuscate(estring(`str`), `counter`)

  result = getAst(genStuff(encodedStr, encodedCounter))
  encodedCounter = (encodedCounter *% 16777619) and 0x7FFFFFFF
```

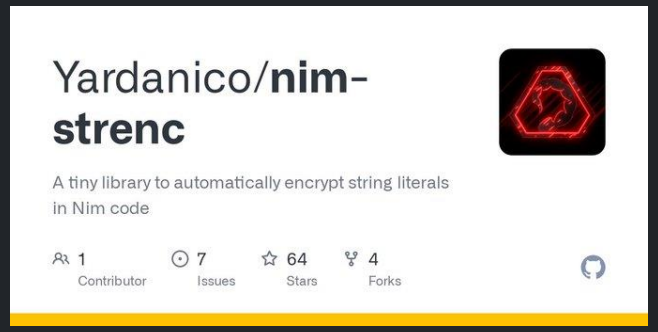


03 | Nim in Practice

Using compile-time macros to obfuscate static strings

```nim
import macros, hashes

type
  estring = distinct string

proc obfuscate(s: estring, key: int): string {.noinline.} =
  var k = key
  result = string(s)
  for i in 0 ..< result.len:
    for f in [0, 8, 16, 24]:
      result[i] = chr(uint8(result[i]) xor uint8((k shr f) and 0xFF))
    k = k +% 1

var encodedCounter {.compileTime.} = hash(CompileTime & CompileDate) and 0x7FFFFFFF

macro encrypt*{s}(s: string{lit}): untyped =
  var encodedStr = obfuscate(estring($s), encodedCounter)

  template genStuff(str, counter: untyped): untyped =
    {.noRewrite.}:
      obfuscate(estring(`str`), `counter`)

  result = getAst(genStuff(encodedStr, encodedCounter))
  encodedCounter = (encodedCounter *% 16777619) and 0x7FFFFFFF
```

Define encryption function (XOR)

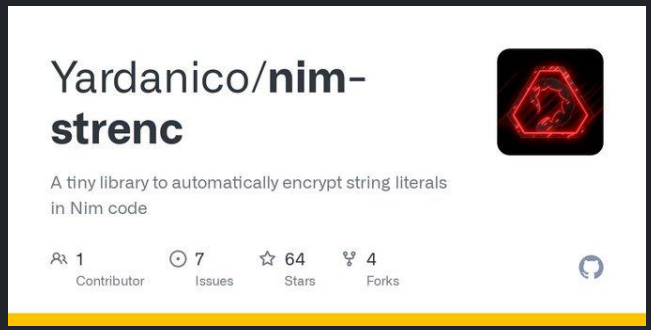


03 | Nim in Practice

Using compile-time macros to obfuscate static strings

```
import macros, hashes

type
  estring = distinct string

proc obfuscate(s: estring, key: int): string {.noinline.} =
  var k = key
  result = string(s)
  for i in 0 ..< result.len:
    for f in [0, 8, 16, 24]:
      result[i] = chr(uint8(result[i]) xor uint8((k shr f) and 0xFF))
    k = k +% 1

var encodedCounter {.compileTime.} = hash(CompileTime & CompileDate) and 0x7FFFFFFF

macro encrypt*{s}(s: string{lit}): untyped =
  var encodedStr = obfuscate(estring($s), encodedCounter)

  template genStuff(str, counter: untyped): untyped =
    {.noRewrite.}:
      obfuscate(estring(`str`), `counter`)

  result = getAst(genStuff(encodedStr, encodedCounter))
  encodedCounter = (encodedCounter *% 16777619) and 0x7FFFFFFF
```

Generate unique key for each string

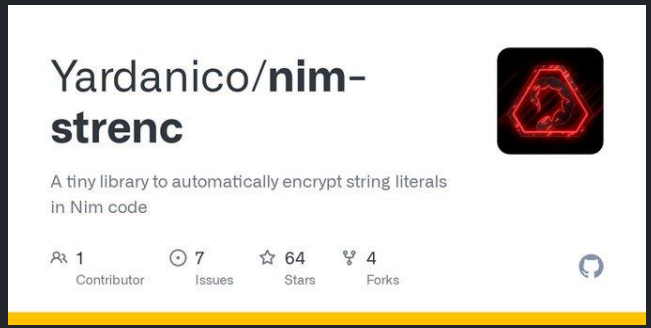


03 | Nim in Practice

Using compile-time macros to obfuscate static strings

```nim
import macros, hashes

type
  estring = distinct string

proc obfuscate(s: estring, key: int): string {.noinline.} =
  var k = key
  result = string(s)
  for i in 0 ..< result.len:
    for f in [0, 8, 16, 24]:
      result[i] = chr(uint8(result[i]) xor uint8((k shr f) and 0xFF))
    k = k +% 1

var encodedCounter {.compileTime.} = hash(CompileTime & CompileDate) and 0x7FFFFFFF

macro encrypt*{s}(s: string{lit}): untyped =
  var encodedStr = obfuscate(estring($s), encodedCounter)

  template genStuff(str, counter: untyped): untyped =
    {.noRewrite.}:
      obfuscate(estring(`str`), `counter`)

  result = getAst(genStuff(encodedStr, encodedCounter))
  encodedCounter = (encodedCounter *% 16777619) and 0x7FFFFFFF
```

Define term-rewriting macro to replace string literals with ‘obfuscate’ function and key

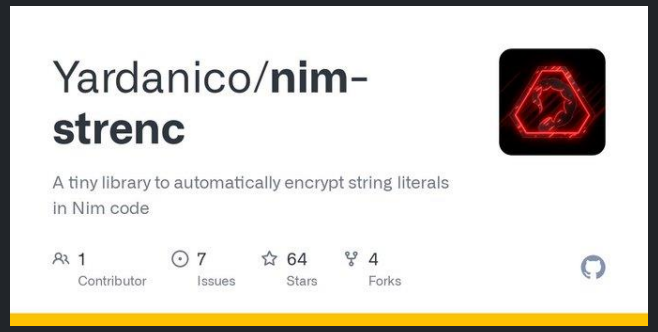


03 | Nim in Practice

Using compile-time macros to obfuscate static strings

```nim
import macros, hashes

type
  estring = distinct string

proc obfuscate(s: estring, key: int): string {.noinline.} =
  var k = key
  result = string(s)
  for i in 0 ..< result.len:
    for f in [0, 8, 16, 24]:
      result[i] = chr(uint8(result[i]) xor uint8((k shr f) and 0xFF))
    k = k +% 1

var encodedCounter {.compileTime.} = hash(CompileTime & CompileDate) and 0x7FFFFFFF

macro encrypt*{s}(s: string{lit}): untyped =
  var encodedStr = obfuscate(estring($s), encodedCounter)

  template genStuff(str, counter: untyped): untyped =
    {.noRewrite.}:
      obfuscate(estring(`str`), `counter`)

  result = getAst(genStuff(encodedStr, encodedCounter))
  encodedCounter = (encodedCounter *% 16777619) and 0x7FFFFFFF
```

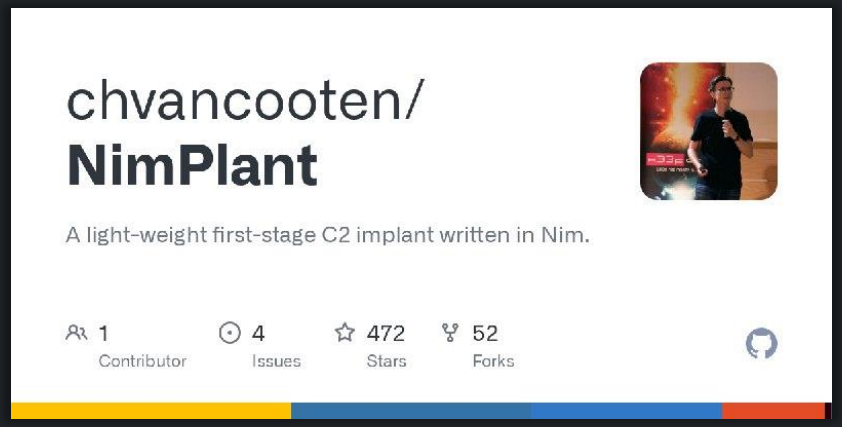
Shift the key



04 | Getting Hands-On

Nimplant: A lightweight stage-one C2

- C2 implant in Nim, server in Python
- Web GUI in Next.JS
- Designed for early-access operations
- Configurable HTTP C2 behavior
- Less suspicious due to native implementations
- Support for BOFs, inline execute-assembly, dynamic shellcode invocation, and more



04 | Getting Hands-On

Demo time!

05 | Getting Started

You don’t need to be a developer!

05 | Getting Started

Getting started with Nim

- Nim basics: official tutorial or nim-by-example
- MalDev for dummies workshop
- OffensiveNim
- Good follows (/): @byt3bl33d3r, @ShitSecure, @ajpc500, @R0h1rr1m
- Communities: BloodHound Slack (#nim), Nim discord (#security)

05 | Getting Started

Getting started with MalDev in other languages

- MalDev for dummies workshop (also C# & Go)
- Sektor7's Malware Development courses (C++)
- Zero-Point security courses (C#)
- Offsec's OSEP (C#/PS/VBA/JS/...)
- 0xPat's blog series (C++)
- Much, much more! Google is your friend