BYOT:
Build Your Own Tools for Fun and Profit

Cas van Cooten

X33fcon Conference
Gdynia, July 22nd, 2022

00 | About

```
[cas@x33fcon ~]$ 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

chvancooten

/in/chvancooten

00 | About

```
[cas@x33fcon ~]$ whoamin’t
```

01 | Offensive Development

The Red Team ❤ their tools

- Red Teaming can be
  - Hard
  - Risky
  - Repetitive

01 | Offensive Development

The Red Team ♥ their tools

- Red Teaming can be
  - Hard -> we collect, analyze, and learn
  - Risky -> we verify
  - Repetitive -> we automate
- Tooling can help with this!
- Knowing your tools (and how to use them) can make or break a good RT operator

GREAT OFFENSIVE TOOLS FROM GITHUB
YOU (PROBABLY)
THE BLUE TEAM

01 | Offensive Development

`To develop or not to develop, that is the question`

Developing in-house is ultimately a business decision

Develop

Build your own tools from scratch

Adapt

Modify open-source projects to fit your needs

Purchase

Purchase operations-ready commercial tools

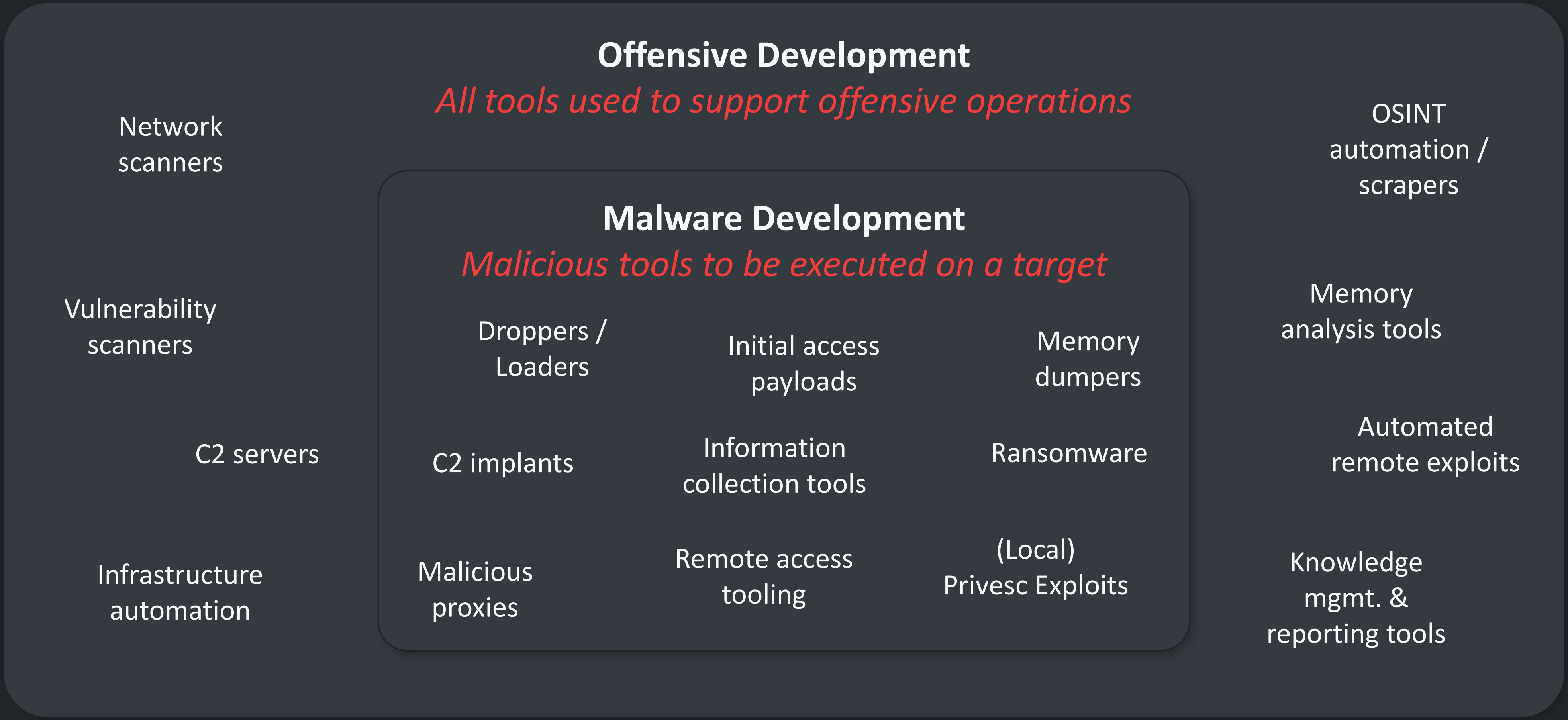
01 | Offensive Development

BYOT means “Build Your Own Tools”

- Development is hard and requires a significant time investment, BUT:
  - It gives you full control over TTPs / IOCs
  - It helps with defense evasion
  - It’s a great learning experience
  - It’s fun!

01 | Offensive Development

“Offensive Development” vs. “Malware Development”



Offensive Development

All tools used to support offensive operations

Network scanners

Vulnerability scanners

C2 servers

Infrastructure automation

OSINT automation / scrapers

Memory analysis tools

Automated remote exploits

Knowledge mgmt. & reporting tools

Malware Development

Malicious tools to be executed on a target

Droppers / Loaders

Initial access payloads

Memory dumpers

C2 implants

Information collection tools

Ransomware

Malicious proxies

Remote access tooling

(Local) Privesc Exploits

...and much more

02 | Malware Development

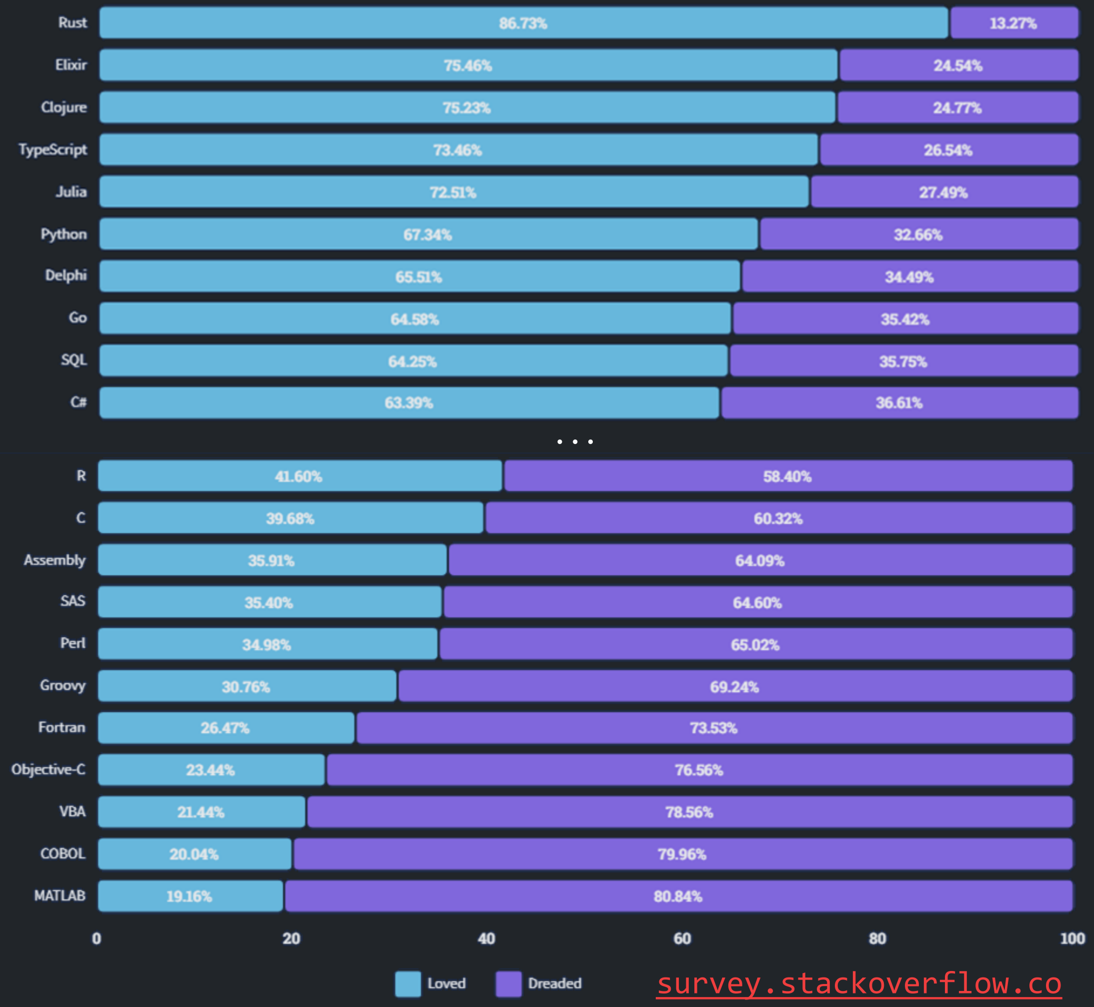
Isn’t malware bad?!

- “Malicious Software”
- To defend against the bad guys, we should think like the bad guys (insert Sun Tzu quote here)
- Defenses are maturing, so we are forced to keep up

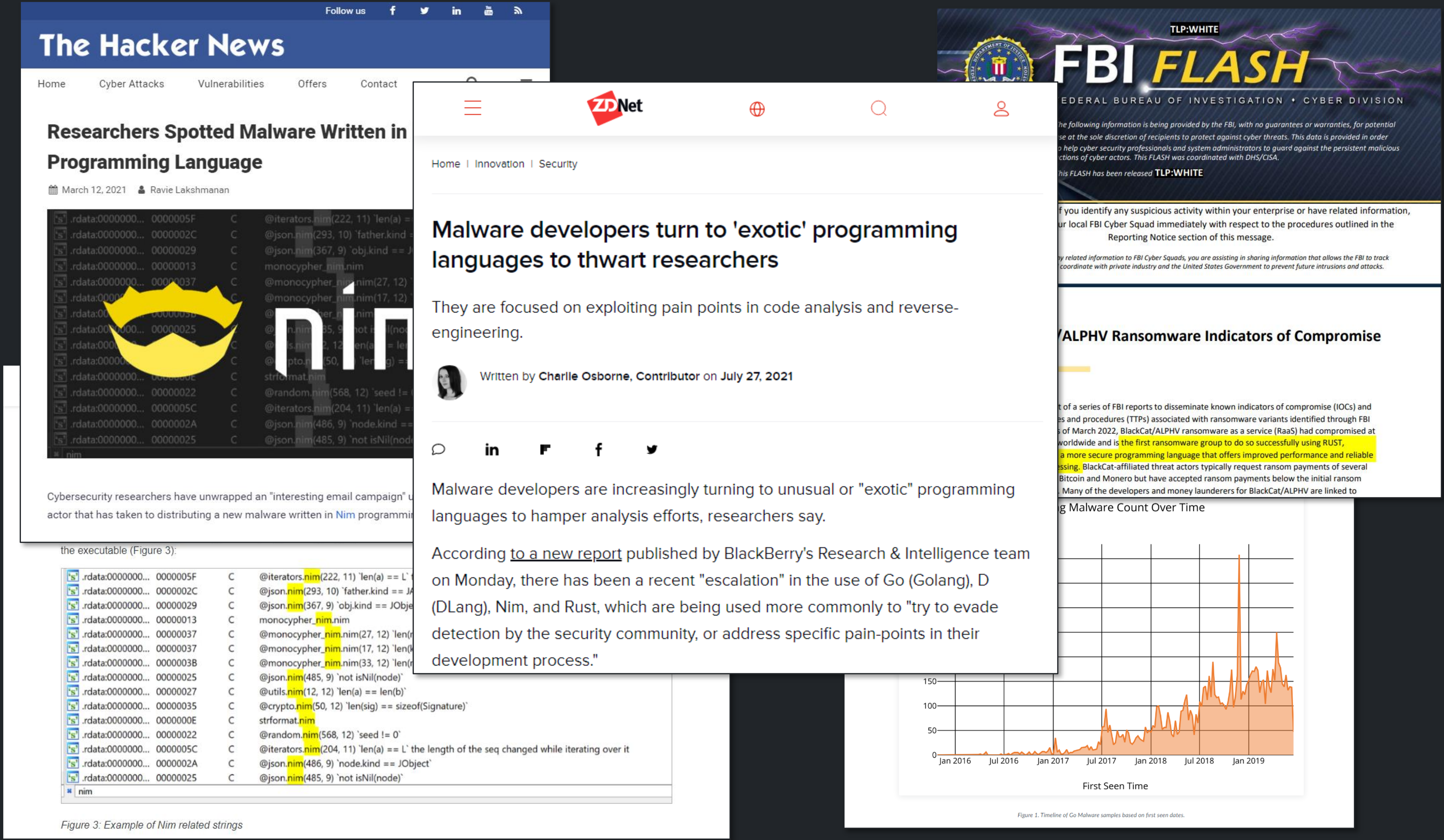
02 | Malware Development

Choosing the right programming language

- Many programming languages can be used, each with benefits and drawbacks
- Considerations:
    - High or low level
    - Interpreted or compiled
    - Developer experience (including docs)
    - Prevalence



survey.stackoverflow.co



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

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 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 ssing. 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.

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(
.rdata:0000000...  00000037  C  @monocypher_nim.nim(17, 12) `len(
.rdata:0000000...  0000003B  C  @monocypher_nim.nim(33, 12) `len(
.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

02 | Malware Development

Digital linguistics

03 | Defense Evasion

The adversary’s adversaries

In a real scenario, you are up against many layers of defenses

Antivirus (AV)

- The most basic defense, but not to be underestimated
- Mostly looks at files statically
- Sometimes uses a sandbox to inspect basic behavior
- Blocks shady stuff

Enterprise Detection and Response (EDR)

- AV on steroids
- Usually uses advanced behavioral detections
- ‘Hooks’ APIs and scans memory for indicators
- Does not always block, may ‘only’ alert!

The Blue Team

- One alert can be enough to ruin your operation
- May dissect your malware to find out more about you
- Will ruin your day

... many others

- Threat hunting
- Other endpoint-based controls
- Network-based controls
- Behavioral analytics
- ...

03 | Defense Evasion

Defensive decision-making

Evasion is effectively a combination of the below (and a bit of luck)

Avoid

- Avoiding locations or activities that are under defensive scrutiny
- E.g. proxying tools rather than executing on a victim endpoint

Blend In

- Making telemetry generated by your malware look as legitimate as possible
- Also involves making clever use of defensive ‘blind spots’!

Sabotage

- Tampering to disrupt the data flow used for defensive purposes
- E.g. patching AMSI/ETW or unhooking function calls

04 | Case Study I

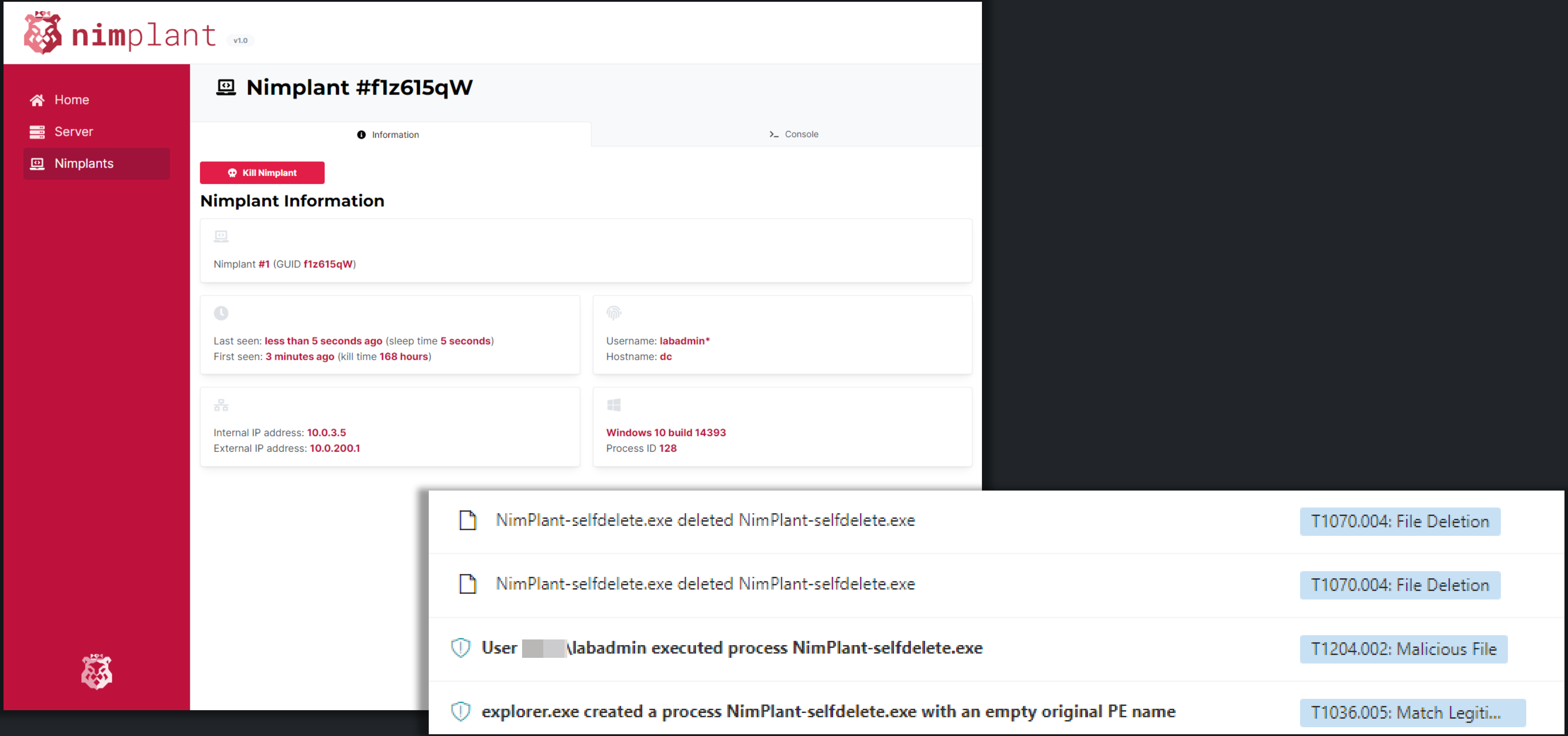
Nimplant: A lightweight stage-one C2

- C2 implant in Nim, server in Python
- Designed for early-access operations
- Focus on evasion by ‘blending in’
- Dangerous functionality compiled into implant separately
- Lesson learnt: Think closely about design before blindly starting dev work (though it was a good learning process!)

casvancooten.com/posts/2021/08/building-a-c2-implant-in-nim-considerations-and-lessons-learned

04 | Case Study I

Nimplant: A lightweight stage-one C2



Nimplant #f1z615qW

Nimplant Information

Nimplant #1 (GUID f1z615qW)

Last seen: less than 5 seconds ago (sleep time 5 seconds)
First seen: 3 minutes ago (kill time 168 hours)

Username: labadmin*
Hostname: dc

Internal IP address: 10.0.3.5
External IP address: 10.0.200.1

Windows 10 build 14393
Process ID 128

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...

05 | Case Study II

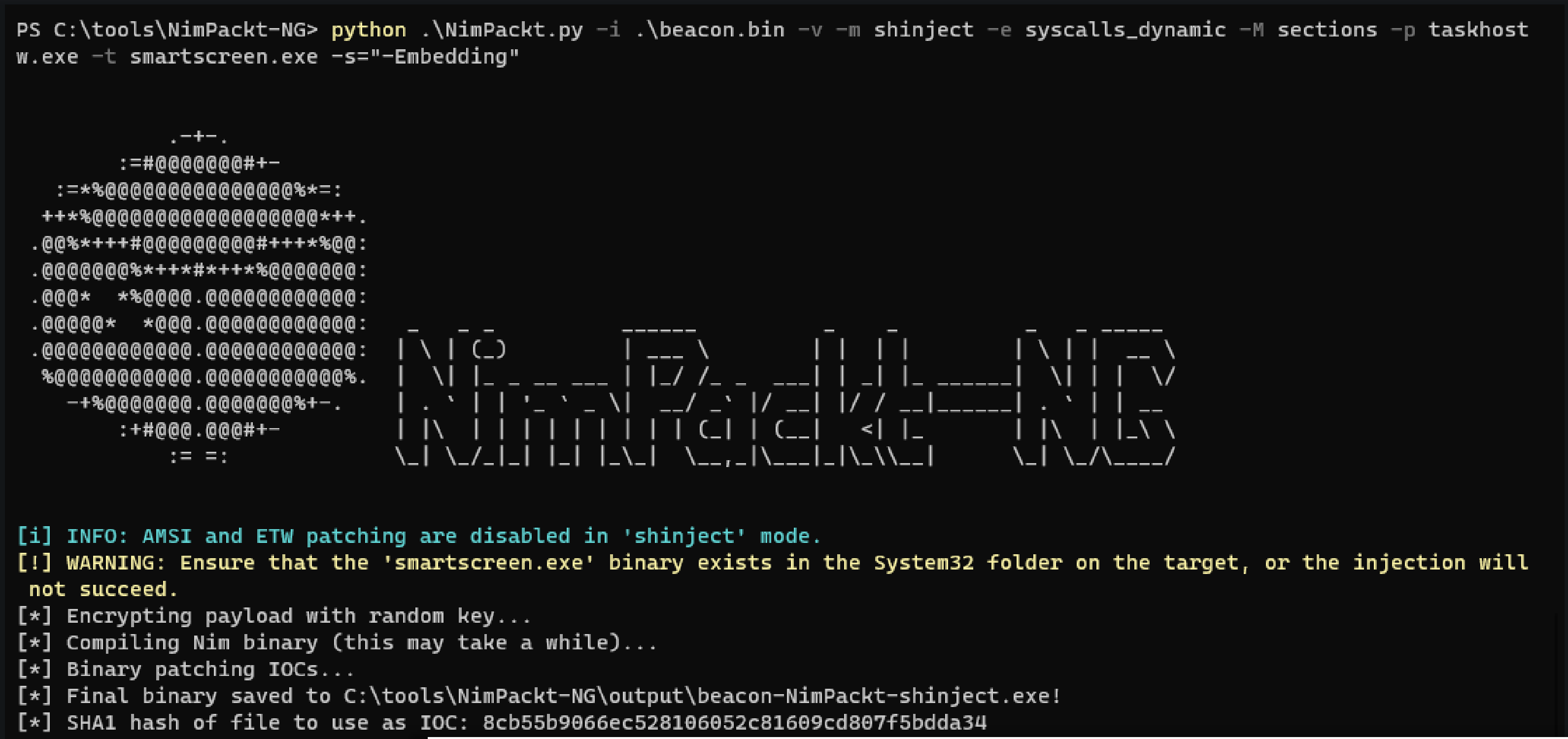
Nimpackt: Shellcode loader and assembly packer

- Packer / loader for .NET assemblies and raw shellcode
- Focus on evasion through ‘sabotage’
- Full re-write after NimPackt-v1
- Lesson learnt: Use the K.I.S.S. principle, design your code to be modular from the start

NIMPACKT

05 | Case Study II

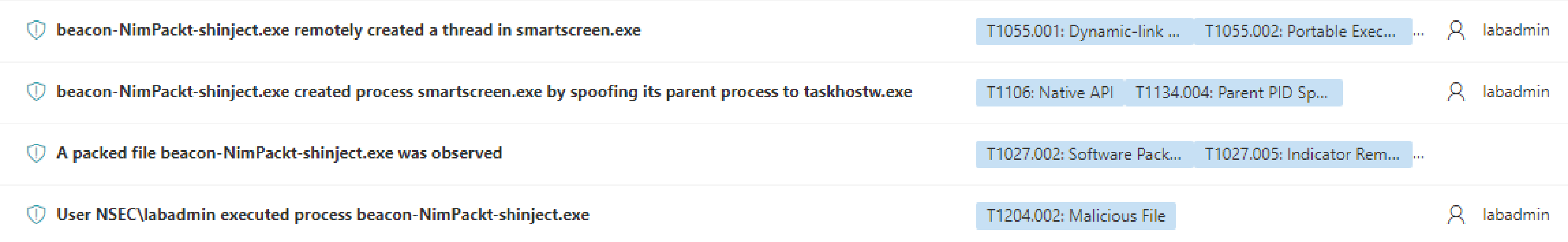
Nimpackt: Shellcode loader and assembly packer



```
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"


            .-+-.
        :=#@@@@@@@#+-
   :=*%@@@@@@@@@@@@@@@%*=:
  ++*%@@@@@@@@@@@@@@@@@@*++.
 .@@%*+++#@@@@@@@@@#+++*%@@:
 .@@@@@@@%*++*#*++*%@@@@@@@:
 .@@@*  *%@@@@.@@@@@@@@@@@@:
 .@@@@@*  *@@@.@@@@@@@@@@@@:   _   _ _           ______          _    _          _   _ _____
 .@@@@@@@@@@@@.@@@@@@@@@@@@:  | \ | (_)          | ___ \        | |  | |        | \ | |  __ \
  %@@@@@@@@@@@.@@@@@@@@@@@%.  |  \| |_ _ __ ___  | |_/ /_ _  ___| | _| |_ ______|  \| | |  \/
    -+%@@@@@@@.@@@@@@@%+-.    | . ` | | '_ ` _ \ |  __/ _` |/ __| |/ / __|______| . ` | | __
        :+#@@@.@@@#+-         | |\  | | | | | | || | | (_| | (__|   <| |_       | |\  | |_\ \
            := =:             \_| \_/_|_| |_| |_|\_|  \__,_|\___|_|\_\\__|      \_| \_/\____/


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

06 | Getting Started

The Offensive Development Mindset

- Ugly code that works > great code that doesn’t
- There is a great community of [offensive|malware] developers
- There are many excellent resources available that you can use as inspiration, cheat sheet, or even “borrow” some code from!
- Note: Never blindly copy-paste! You won’t learn, and open sources are fingerprinted by defensive tools

06 | Getting Started

Resources for getting started

- My MalDev for dummies workshop (C#/Nim)
- Sektor7's Malware Development Essentials course (C++)
- Zero-Point security courses (C#)
- Offsec's OSEP (C#/PS/VBA/JS/...)
- Repo's like OffensiveNim (Nim)
- Countless blogs, video tutorials, and/or books!

See slides at github.com/chvancooten/conferences for clickable links

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
- Detection based on file hash is (near-)worthless

08 | Takeaways

Time to get devvin'!

- Developing, adapting, or purchasing tools are all valid options (and can be combined)
- Development skills will help emulate adversaries and bypass modern defenses
- Every language has its pros and cons when it comes to offensive development
- It doesn't take much to get started!