

GENETIC MALWARE

DESIGNING PAYLOADS FOR
SPECIFIC TARGETS

Who we are

- Travis Morrow
 - AppSec, Mobile, WebTesting, SecOps
- Josh Pitts
 - Author of BDF/BDFProxy
 - <https://github.com/secretsquirrel>
 - AppSec, RedTeaming, WebTesting, SecOps

The background of the slide is a faded, light-colored image. On the right side, the front of a vintage car is visible, showing a large round headlight and a circular grille. On the left side, there is a large, leafy green plant, possibly a fern or a similar foliage. The overall tone is soft and nostalgic.

How we got here...













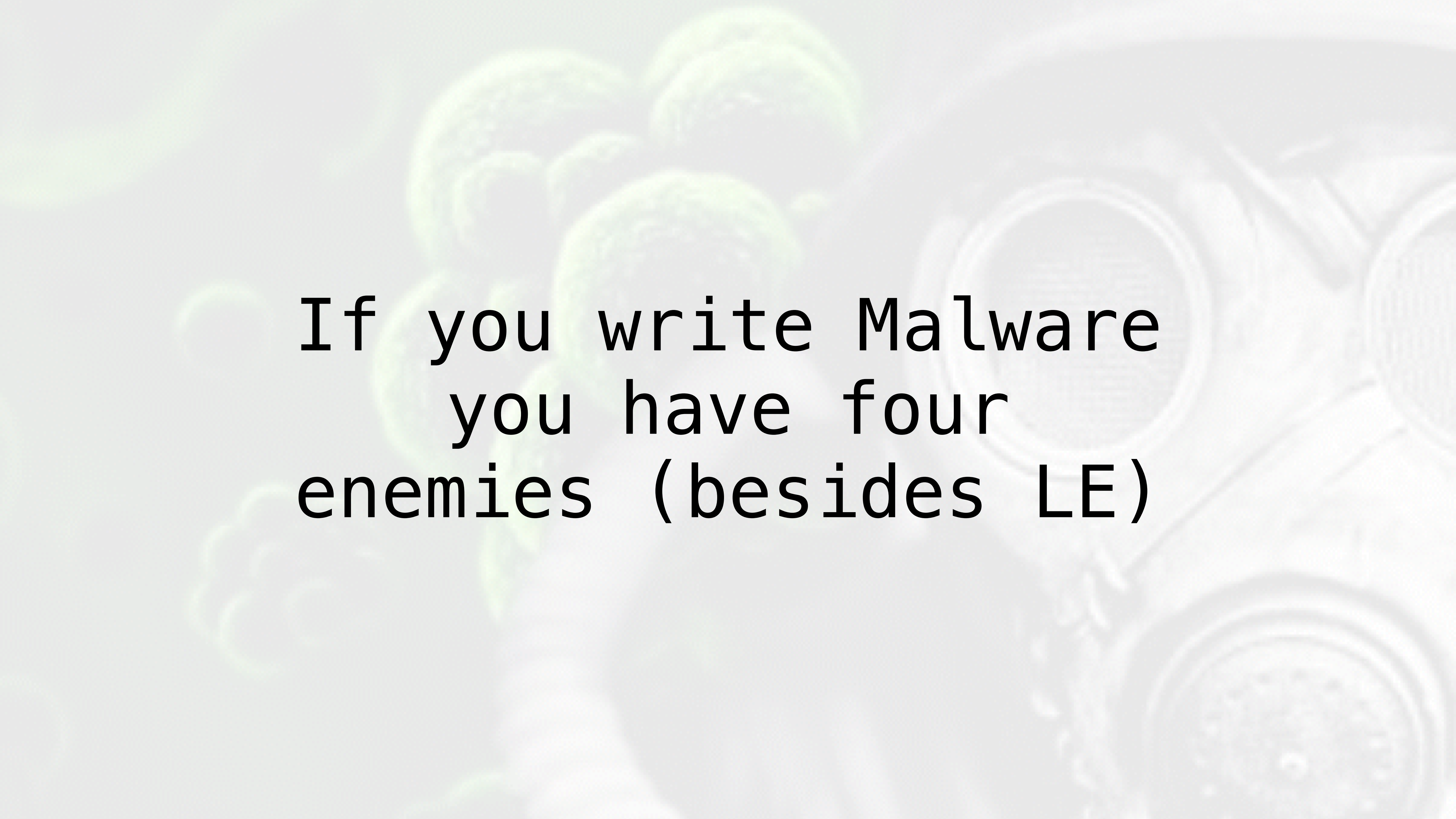
Dude,
I have this algo...





Awesome
Let's do it..





If you write Malware
you have four
enemies (besides LE)

Conduct Operations

If you write ~~Malware~~
you have four
enemies (besides LE)

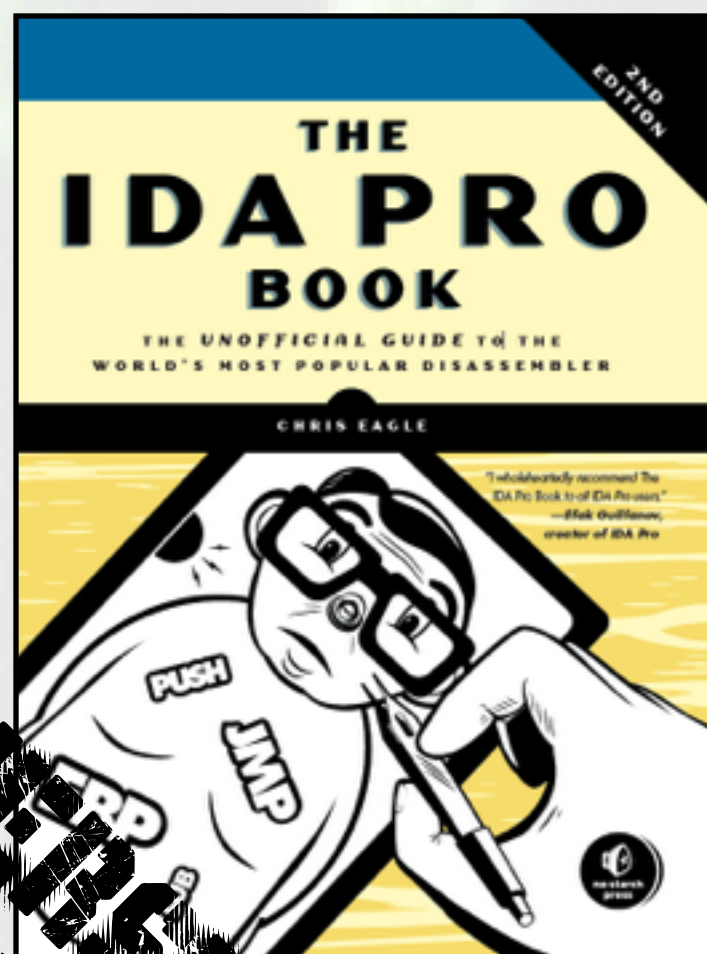
ANTI-VIRUS



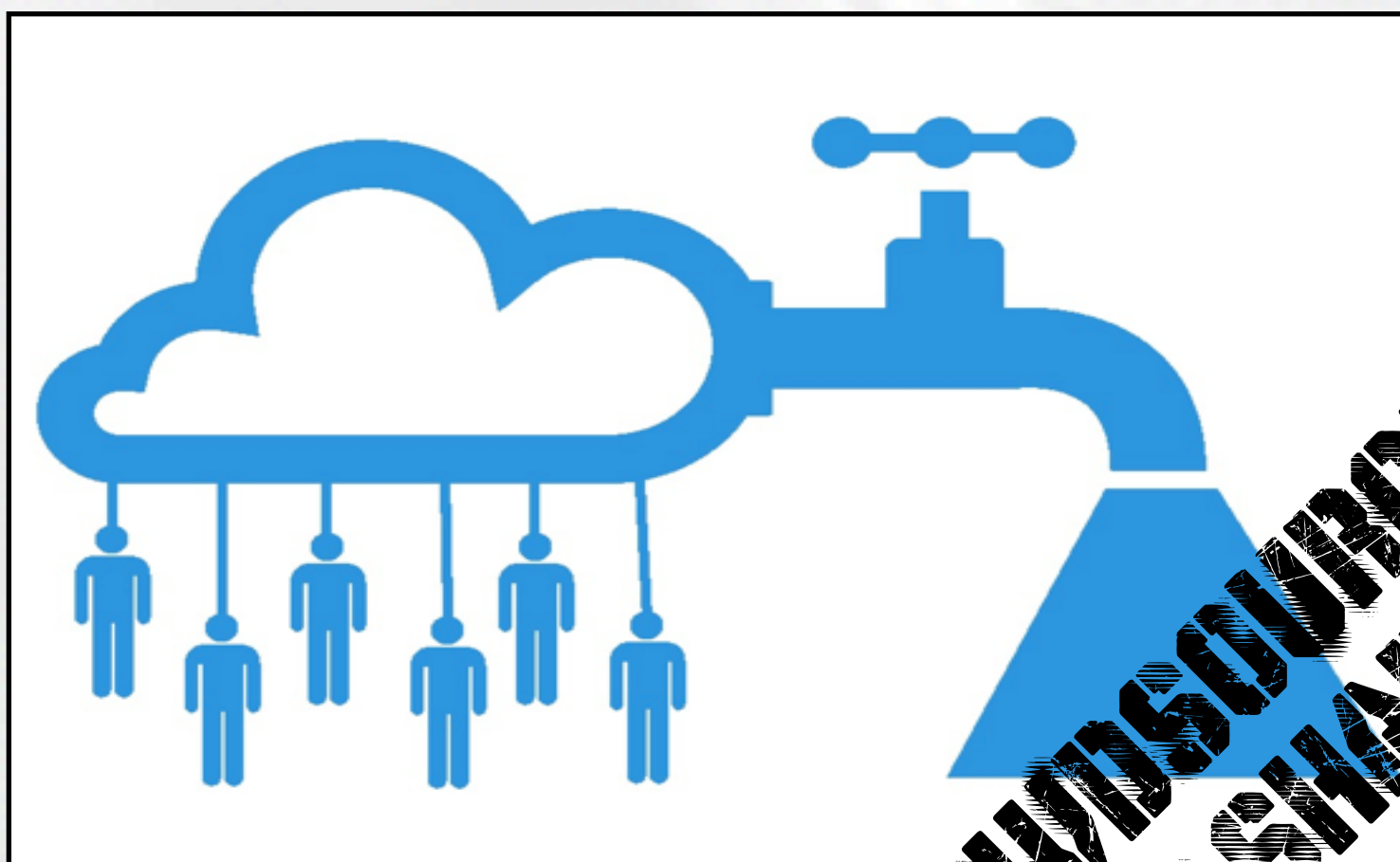
AUTOMATED
SAMPLE TEST



THE IDA PRO
BOOK



COMMON
WE-C-SHARING



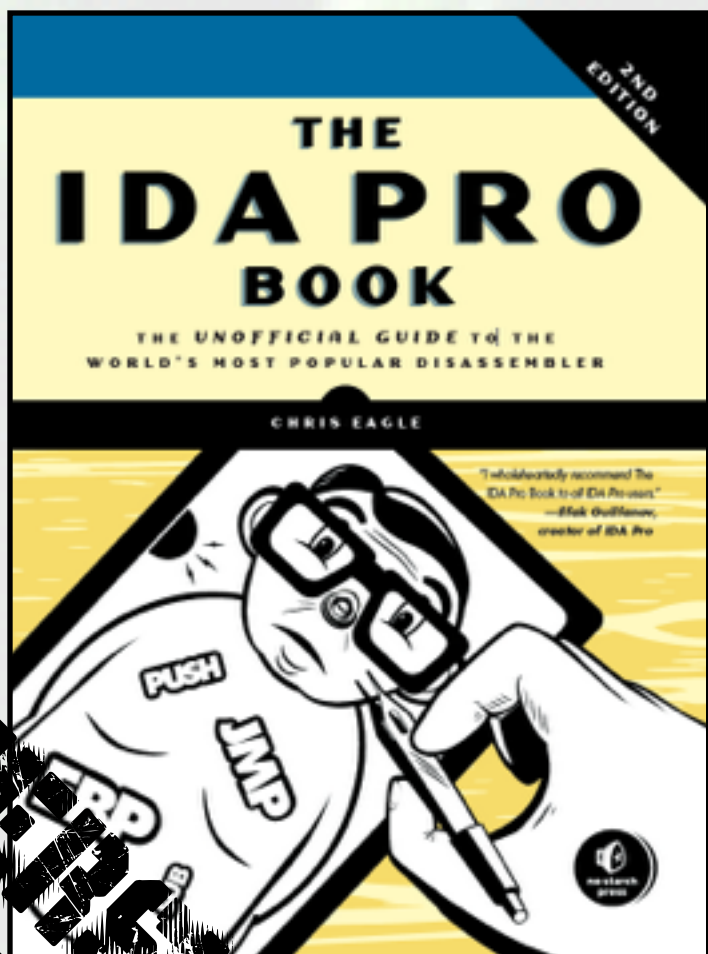
ANTI-VIRUS



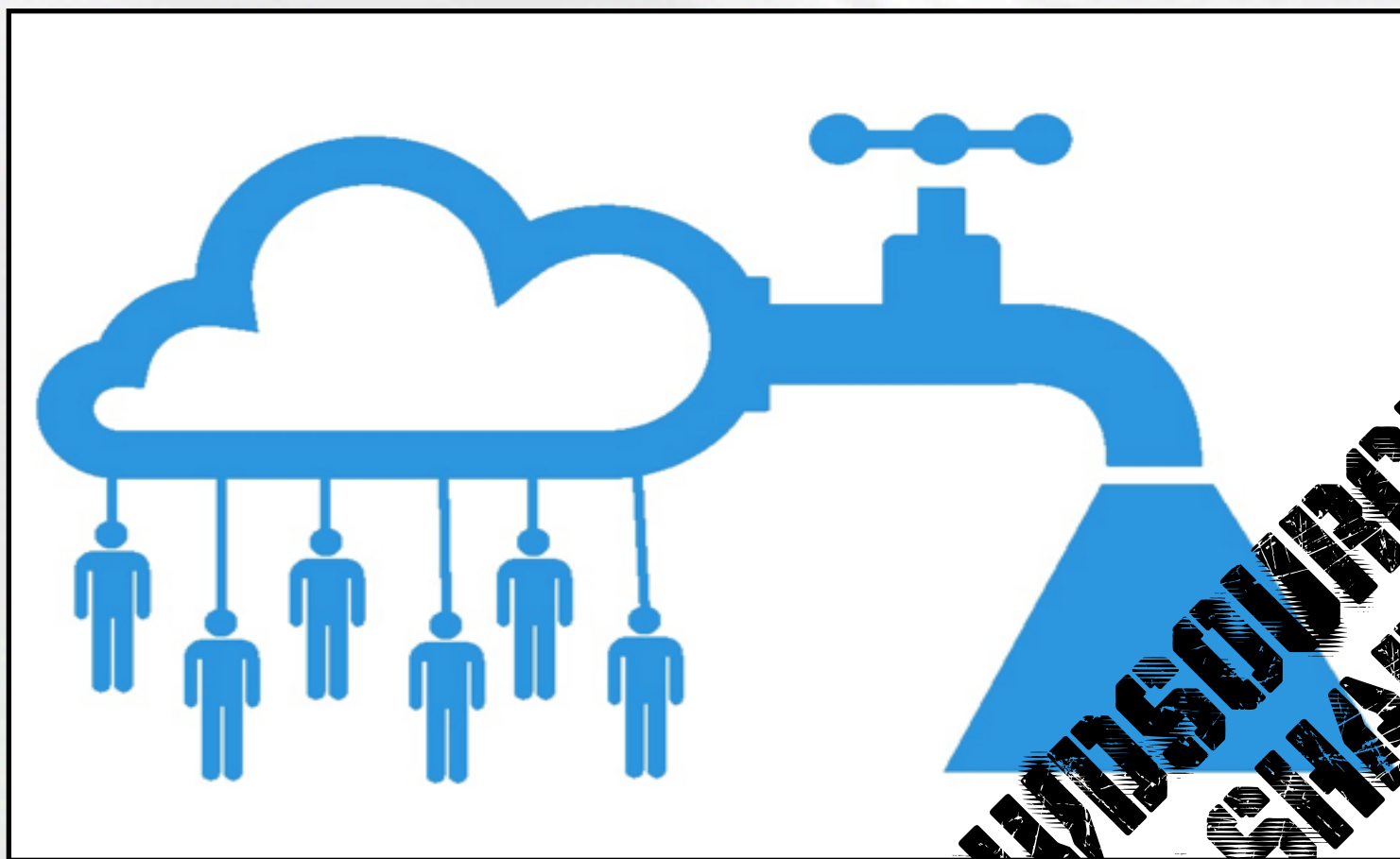
AUTOMATED
SAMPLE TEST



THE IDA PRO
BOOK



COMMON
WE-COMPARING





- Including Consumer Grade Products
- Founded by the Charlie Sheen of our industry
- Easy to bypass, not really a concern
- Can make you more vulnerable
- Respect for F-Secure and Kaspersky



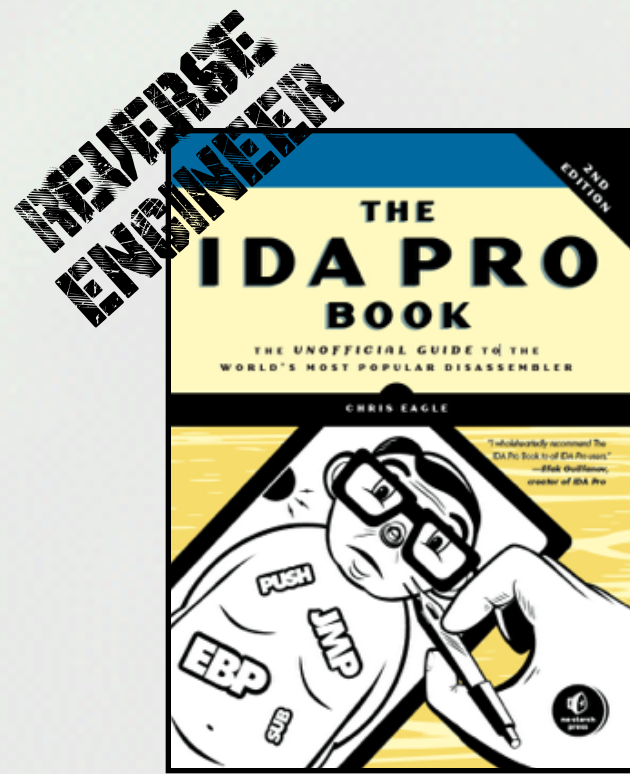
- Including Consumer Grade Products
- Founded by the Charlie Sheen of our industry
- Easy to bypass, not really a concern
- Can make you more vulnerable
- Respect for F-Secure and Kaspersky



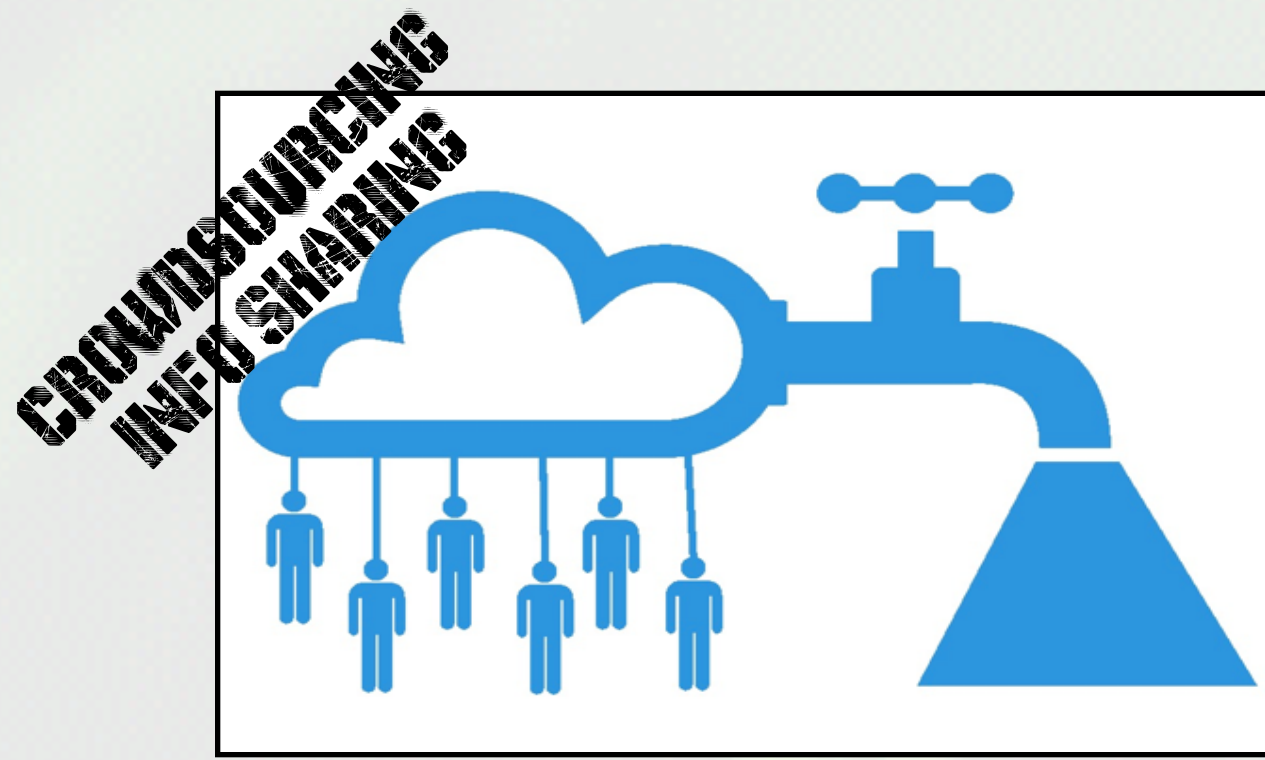
- Easy to bypass analysis
- A lot of machines are still XP
- They often:
 - Have unique ENV vars
 - Rarely change external IP
 - Have analysis timeouts



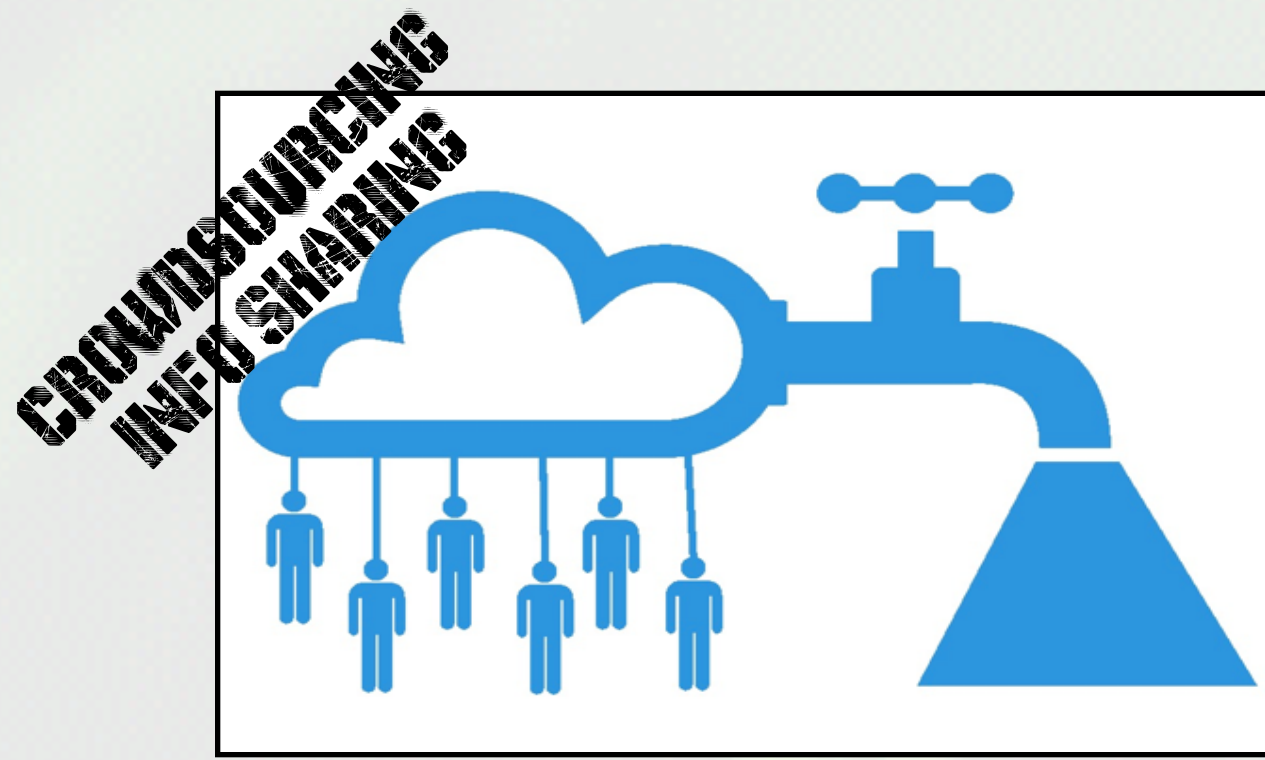
- Easy to bypass analysis
- A lot of machines are still XP
- They often:
 - Have unique ENV vars
 - Rarely change external IP
 - Have analysis timeouts




- Hard to defeat the Reverse Engineer (RE)
- Tricks that defeat AV and Automated Sandboxes != work on an experienced RE
- If malware payloads decrypt in memory on the RE's machine, it can be analyzed
- At best you can only slow down the RE
- Turn RE into a password cracker and you win



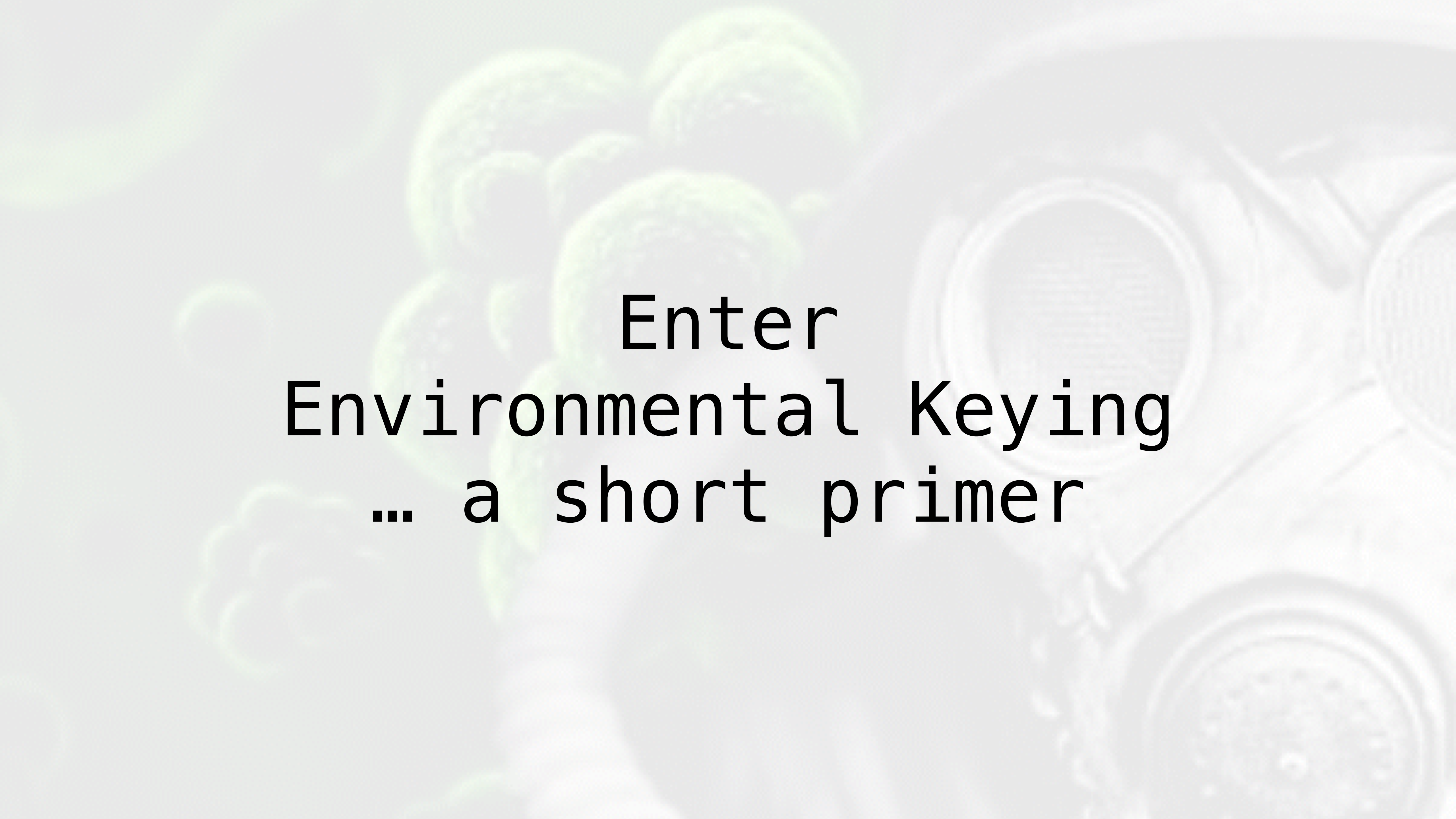
- Kind of a MMO of Whack-A-Mole
- Magnifies the outcome of easy to fingerprint malware
- Defeat the RE and this becomes less effective



- Kind of a MMO of Whack-A-Mole
- Magnifies the outcome of easy to fingerprint malware
- Defeat the RE and this becomes less effective



Enter Environmental Keying



Enter Environmental Keying ... a short primer

Clueless Agents

- Environmental Key Generation towards Clueless Agents (1998) – J. Riordan, B. Schneier
- Several methods for key sources:
 - Server required
 - Usenet
 - Web pages
 - (Forward|Backwards)–Time Hash Function
 - Host specific
 - Mail messages
 - File System
 - Local network

Clueless Agents

- Environmental Key Generation towards Clueless Agents (1995) – J. Riordan, B. Schneier
- Several methods for key sources:
 - Server required
 - Usenet
 - Web pages
 - (Forward|Backwards) Time Hash Function
 - Home network
 - Mail messages
 - File system
 - Local network

Secure Triggers

- Foundations for Secure Triggers (2003), Corelabs
 - Did not reference Clueless Agents
- Defeat REs and analysis
- Makes mention of OTP
- Lots of Math (too much)

Secure Triggers

- Foundations for Secure Triggers 0037, Corelabs
- Did not reference Clueless Agents
- Defeat Rot analysis
- Manipulation of OTP
- Lots of Math (too much)

Bradley Virus

- Strong Cryptography Armoured Computer Virus Forbidding Code Analysis (2004), Eric Filiol
- References Clueless Agents
- Nested encrypted enclaves/payloads
- “Complete source code is not available”
- “[...]cause great concern among the antiviral community. **This is the reason why will not give any detailed code.**

Bradley Virus

- Strong Cryptography Armoured Comput. Virus Forbidding Code Analysis (2014), Eric Filiol
- References Clueless Reports
- Nested encrypted enclaves/payloads
- “Complete source code is not available”
- “[...] caused great concern among the antiviral community. **This is the reason why will not give any detailed code.**

Hash and Decrypt

- Mesh design pattern: hash-and-decrypt (2007), Nate Lawson
- Application of secure triggers to gaming

Hash and Decrypt

- Mesh design pattern: hash-and-decrypt (2007), Mark Lawson
- Application of secure triggers to gaming

Über-Malware

- Malicious Cryptography... Reloaded (CanSecWest 2008) – E.Filiol, F.Raynal
- New: Plausible Deniability!
 - Via OTP
 - POC was a XOR

Über-Malware

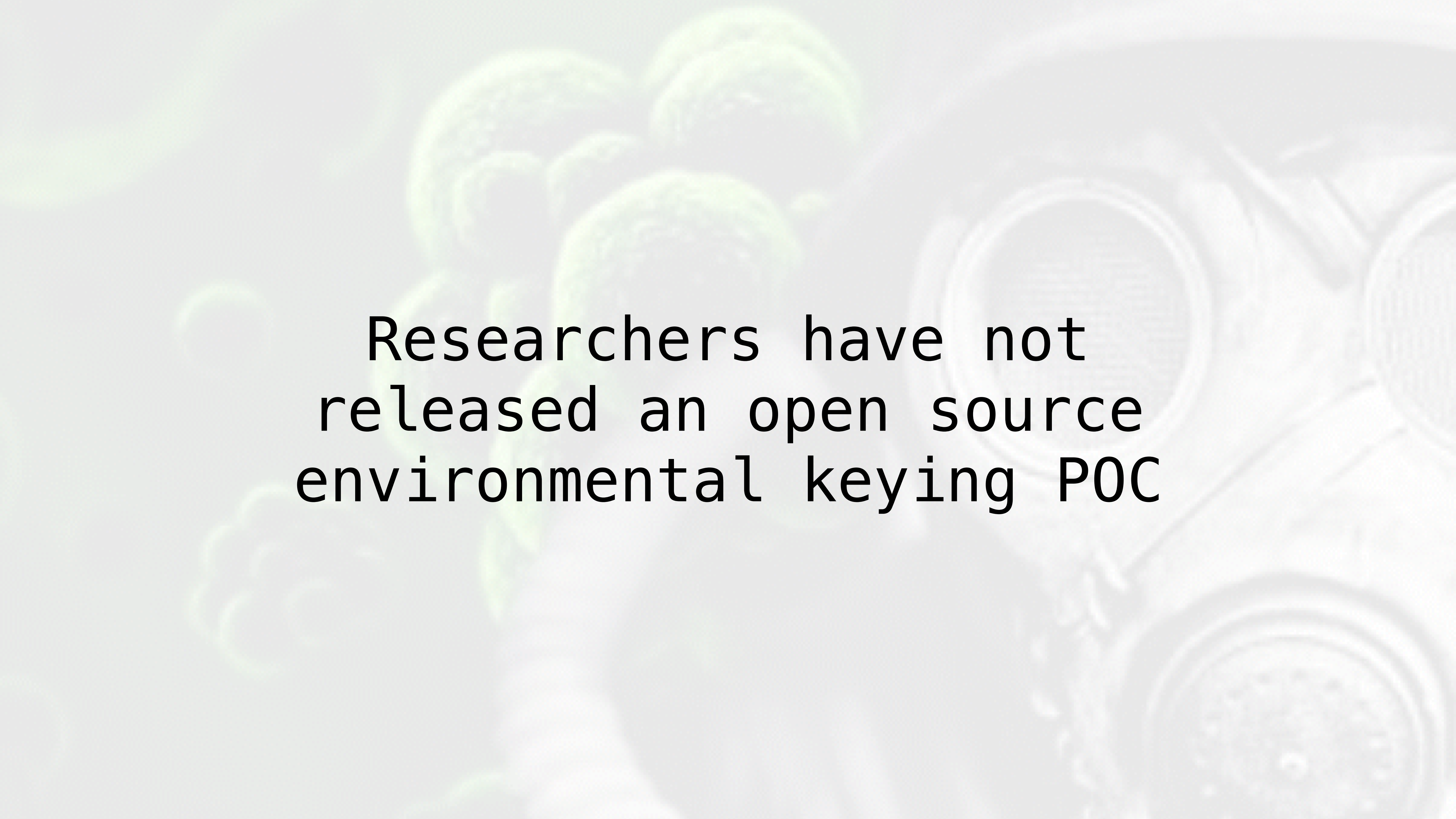
- Malicious Cryptography. Reloaded (CanSecWest 2008) – E.Filiol, F.Raoult
- New: Plaintext Deniability!
- via CFP
- POC was XOR

Impeding Automation

- Impeding Automated Malware Analysis with Environmental-sensitive Malware (2012), Usenix, (C.Song, et al)
 - Did not reference Clueless Agents or the Bradley Virus
- Rediscovered Environmental Keying..
- Examples of Environmental keys
- Great Quotes:
 - “Due to time constraints..”
 - “[...]exceeds the scope of this paper,[...]”
 - “At the inception of this paper, concerns were raised[...]”

Impeding Automation

- Impeding Automated Malware Analysis with Environmental sensitive Malware (2012), Usenix (C.Song, et al)
- Did not reference Clueless Agent or the Bradley Virus
- Rediscovered Environmental Keying.
- Examples of environmental keys
- Great Quotes
 - “Due to some constraints...”
 - “[...] exceeds the scope of this paper, [...]
 - “At the inception of this paper, concerns were raised[...]

The background of the slide is a composite image. On the right side, there is a close-up, slightly blurred view of a car's engine compartment, showing various mechanical parts like the air filter and belts. On the left side, there is a green, leafy plant, possibly a succulent, also slightly blurred. The overall background has a light, desaturated tone.

Researchers have not
released an open source
environmental keying POC



Flashback (2011)

Flashback (2011)

- Mac OS X only malware
- Initial agent sent back UUID of OS to server
- Server used MD5 of UUID to encrypt payload
- Sent back to user and deployed



Gauss (2012)

Gauss (2012)

- Discovered by Kaspersky
- Encrypted Payload “Godel”
- Key derived from directory path in program files, MD5 hashed for 10k rounds
- Not publicly decrypted to date



Targeted Malware Compared to Biological/ Chemical Agents



Chemical Agents

- Area effect weapons
- Effective for days to weeks
- For targeting systems:
 - Domain specific env vars
 - External IP address
 - Check system time

Biological Agents

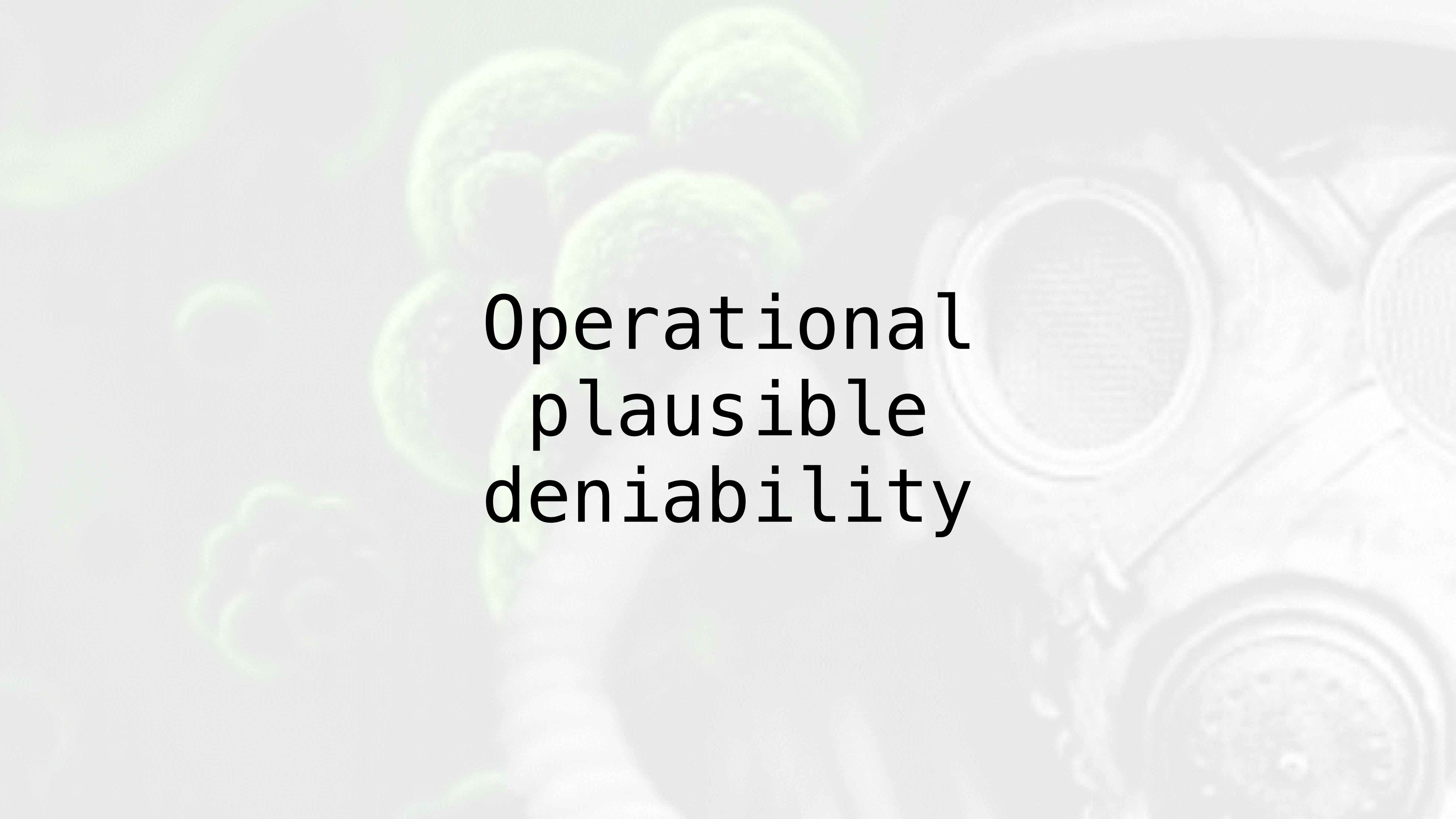
- Viral
- Genetic Targeting
- “Ethnic Weapons”
- For systems targeting:
 - Path
 - Particular file (OTP)

The background of the slide features a close-up, slightly blurred image of a car engine. On the left side, there is a plume of green smoke or steam rising from the engine area. The engine components, including various pipes, hoses, and circular openings, are visible in a light gray tone.

Targeted Malware and its use in Operations



Deploy everywhere
work somewhere

The background of the slide is a composite image. On the right side, there is a close-up, grayscale image of a car engine, showing various mechanical components like the air filter and belts. On the left side, there is a soft-focus image of a green plant with large, rounded leaves. The text is centered over the engine image.

Operational
plausible
deniability

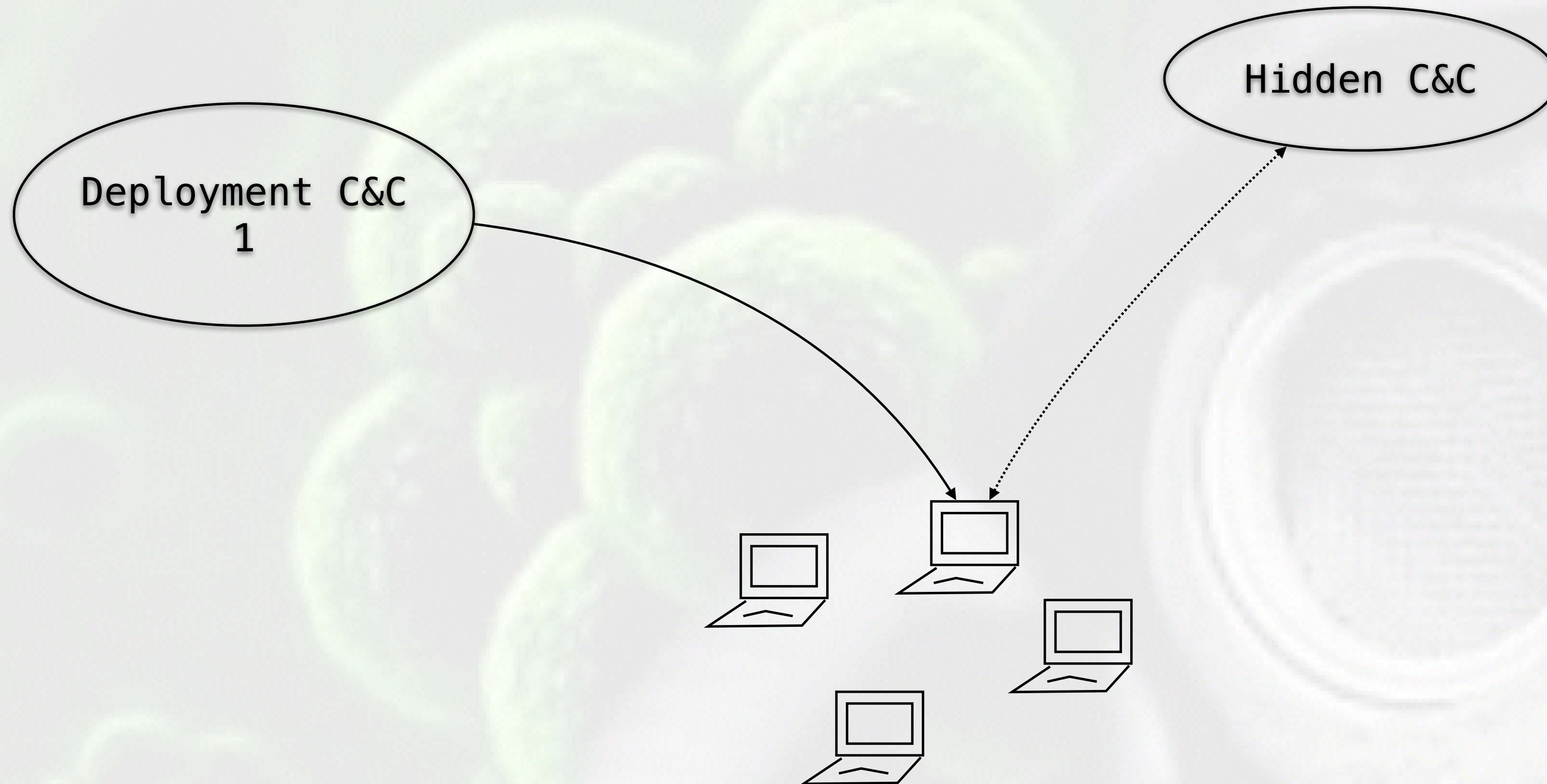
The background of the slide features a faded, light-colored image of a car engine on the right side, showing various mechanical components like the air filter and belts. On the left side, there are green, leafy plants, possibly ferns, which are also faded. The overall background is a light, neutral tone.

Hidden Command and Control (C&C)

Deployment C&C
1

Hidden C&C





Deployment C&C
1

Hidden C&C



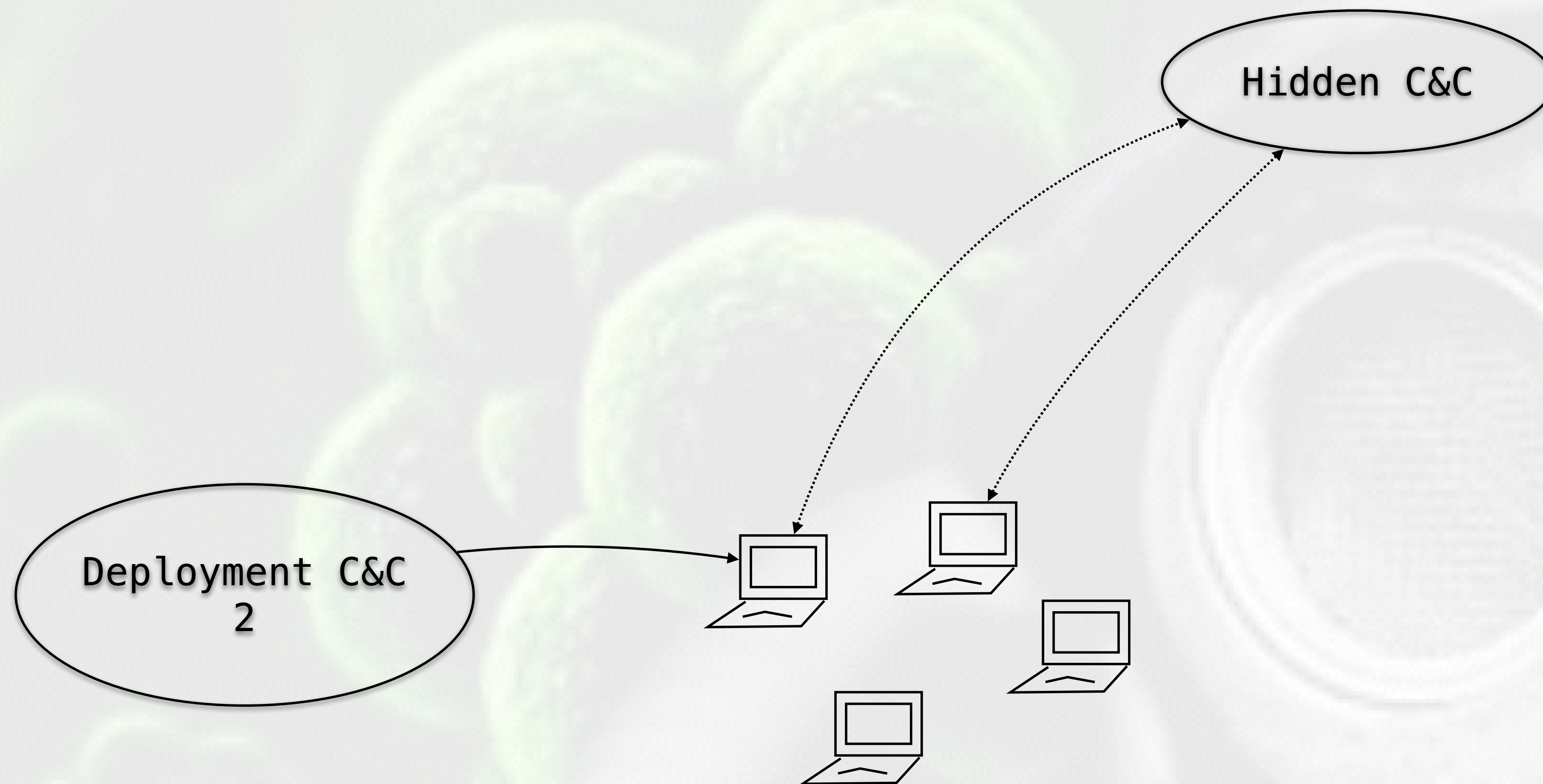
Hidden C&C

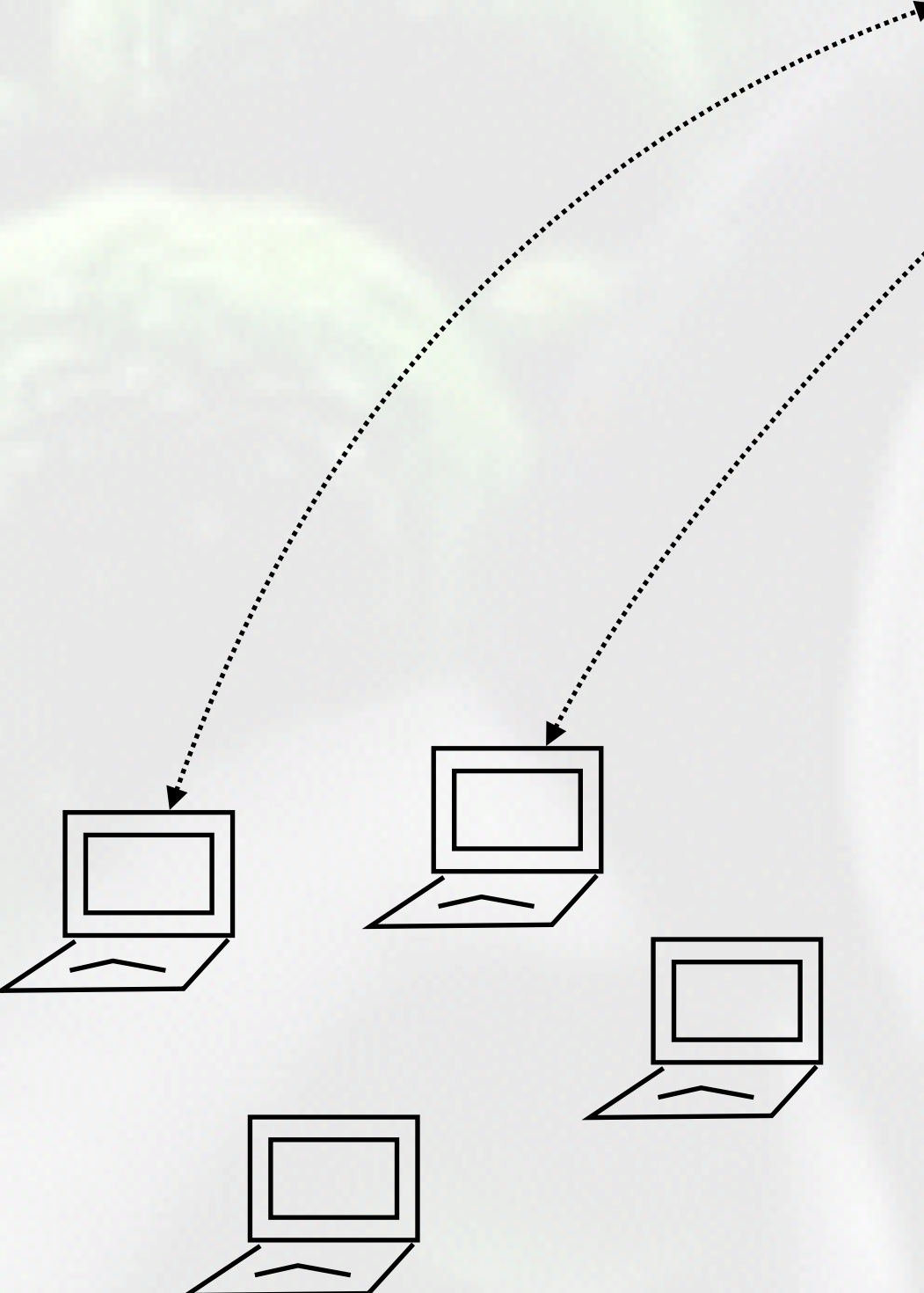
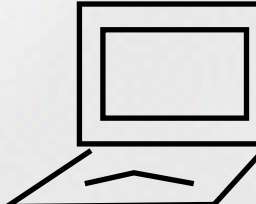
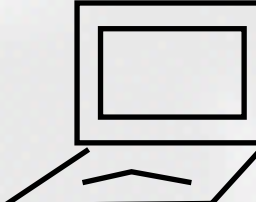
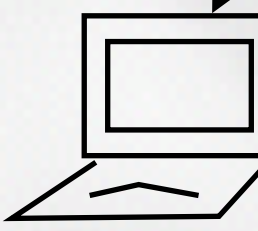
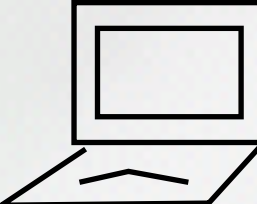
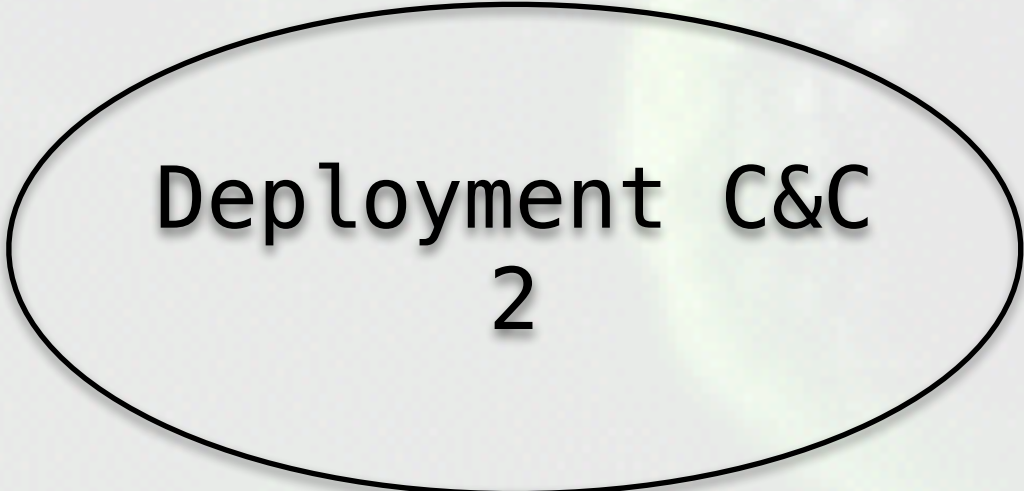


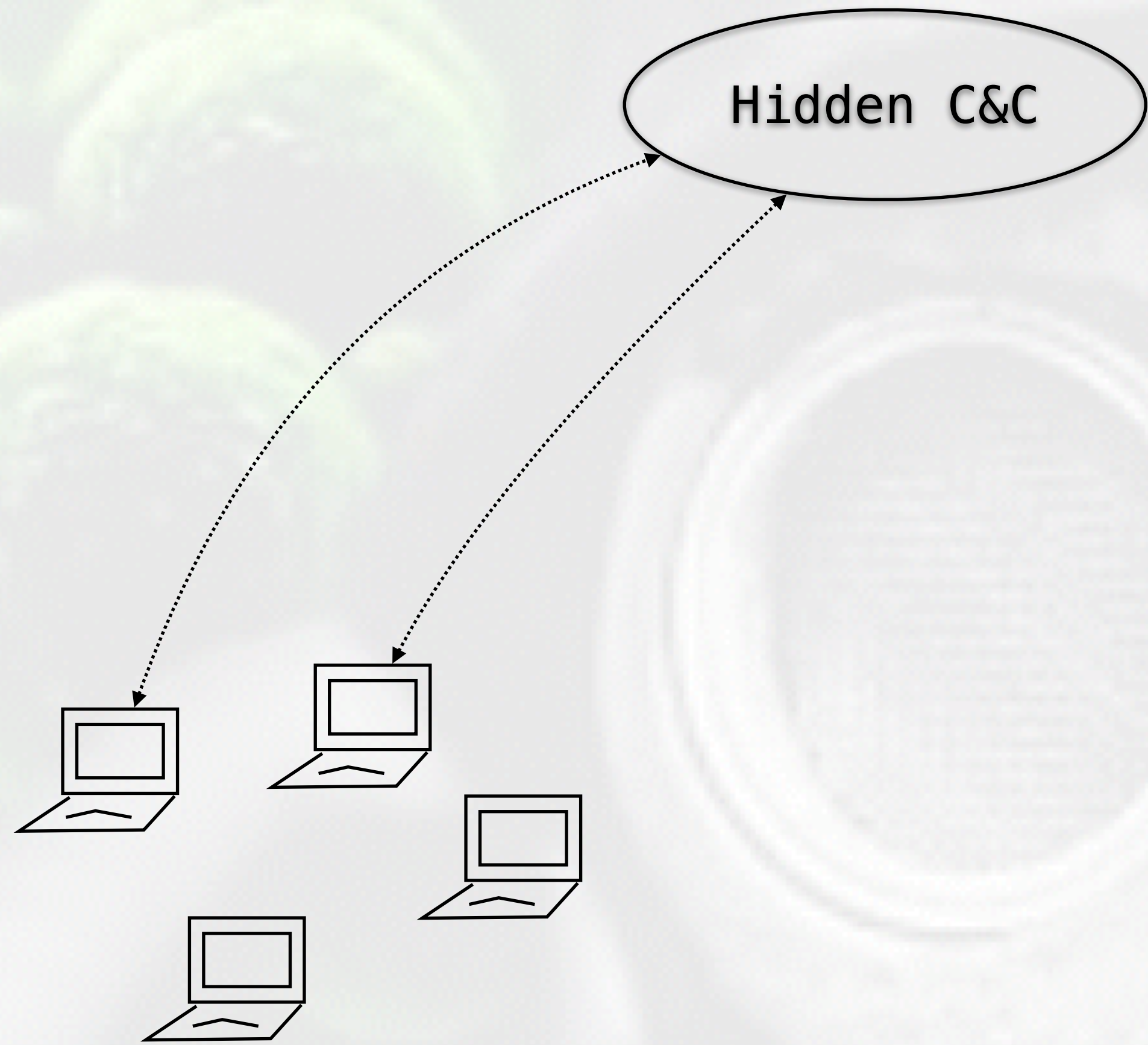
Deployment C&C
2

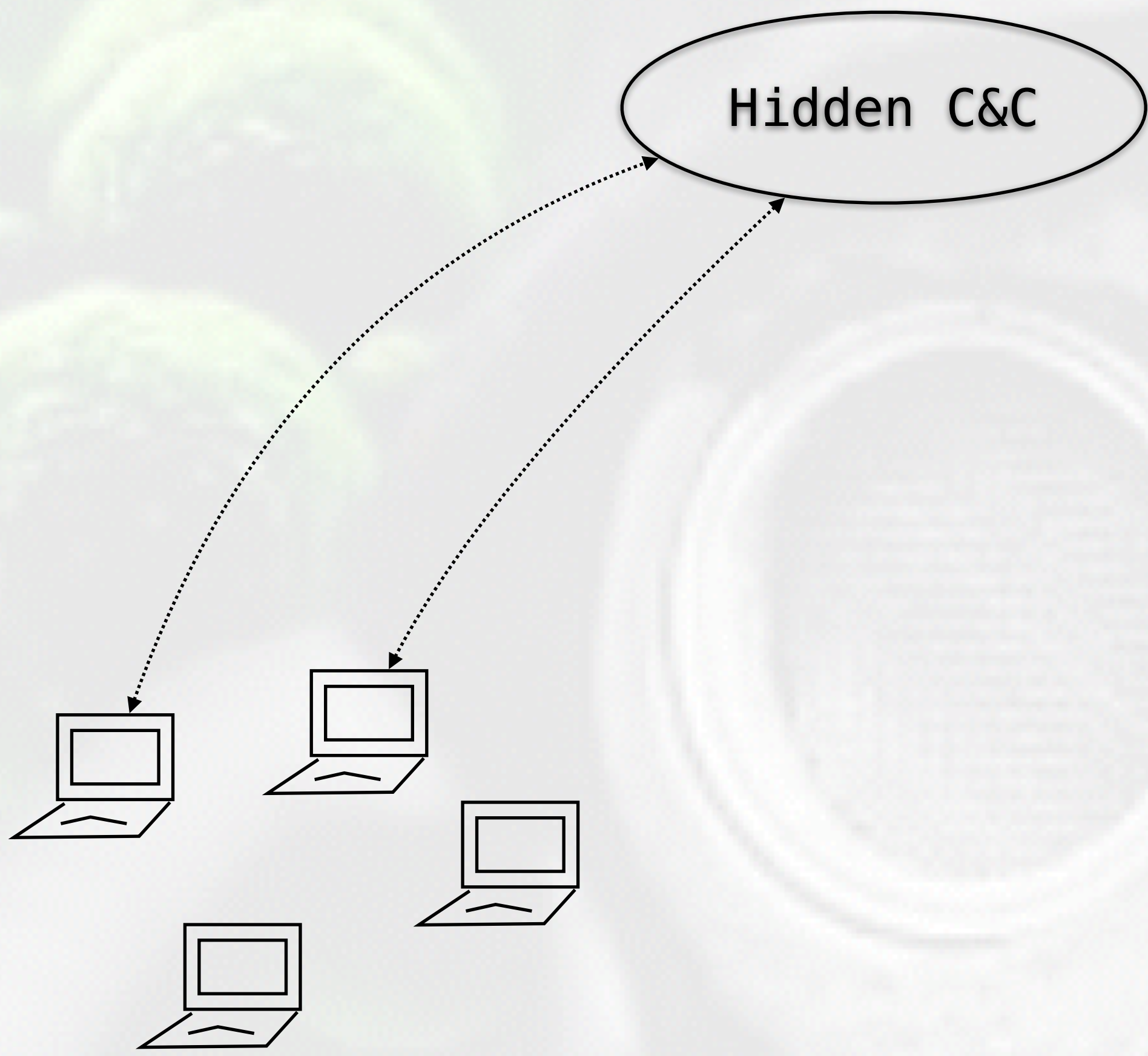
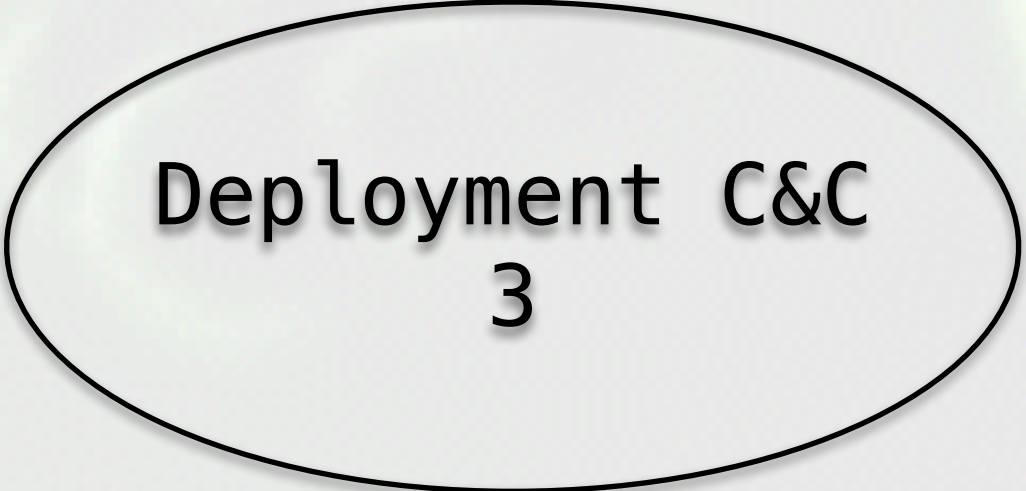


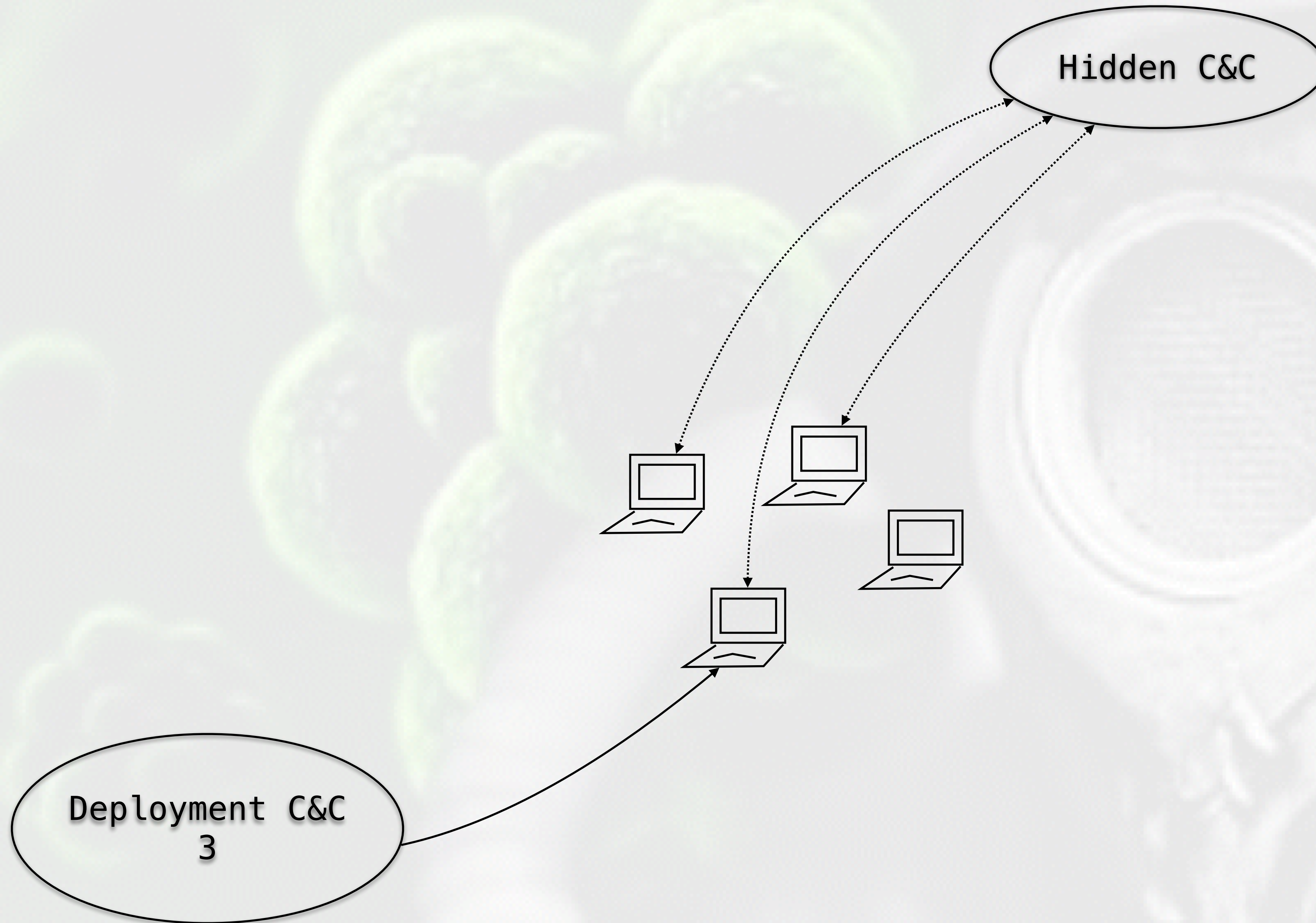
Hidden C&C

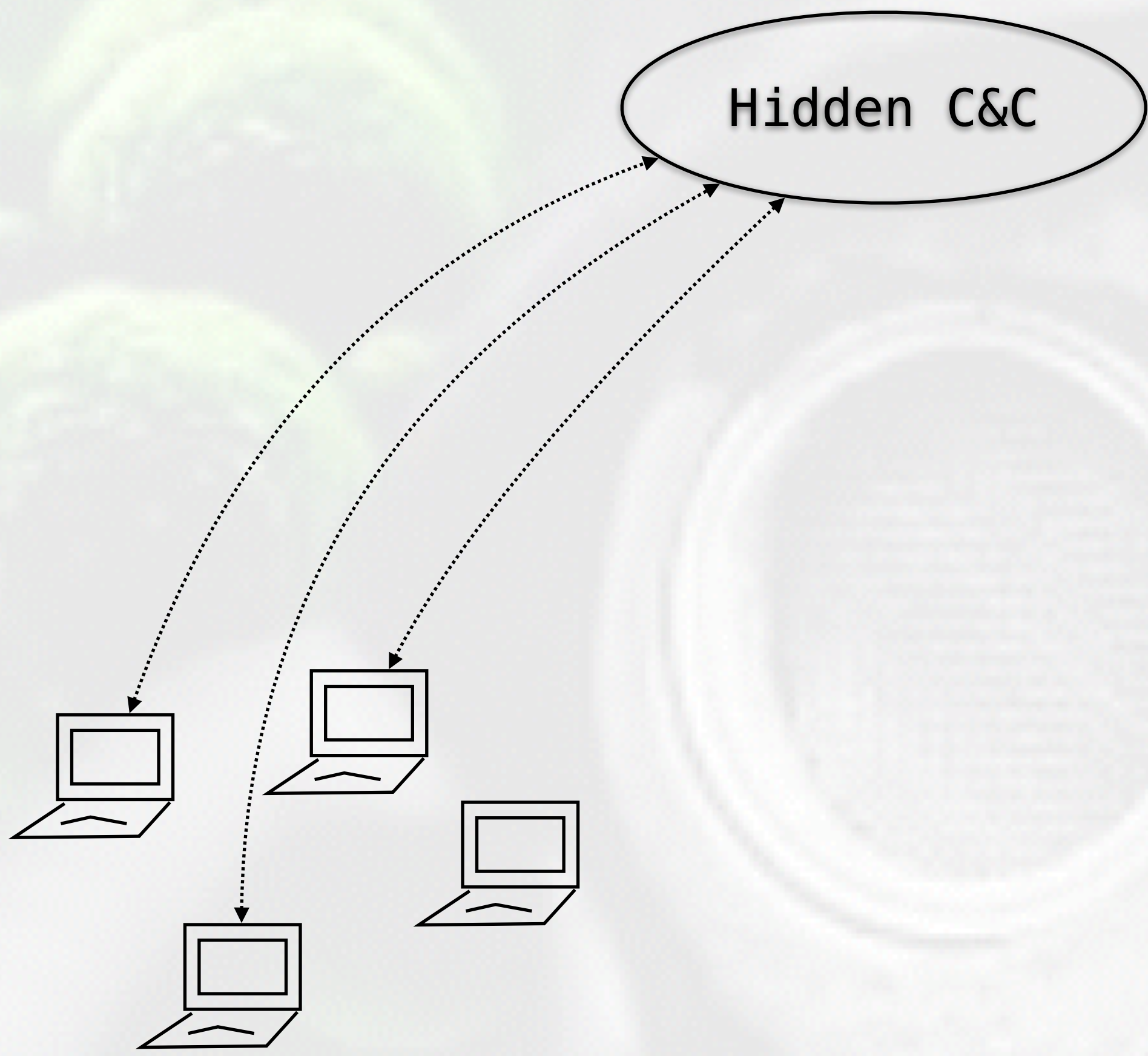
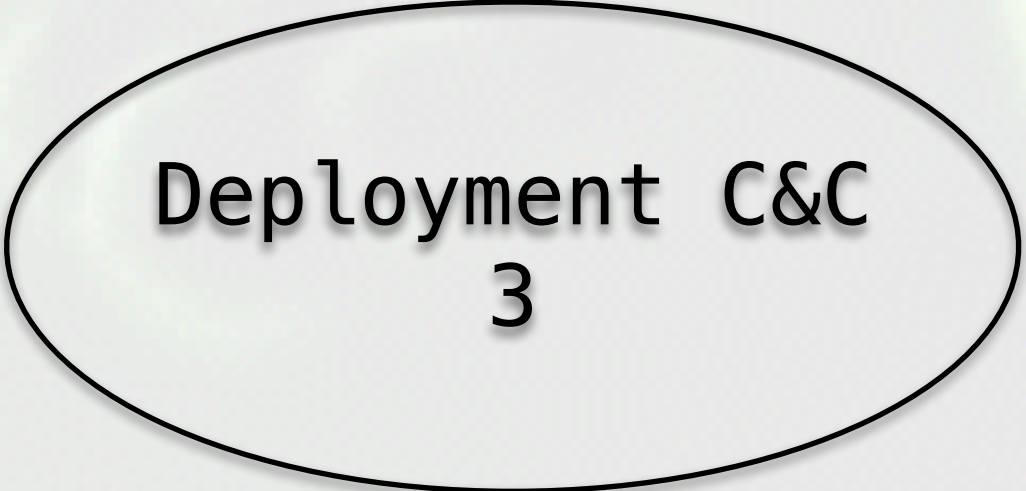


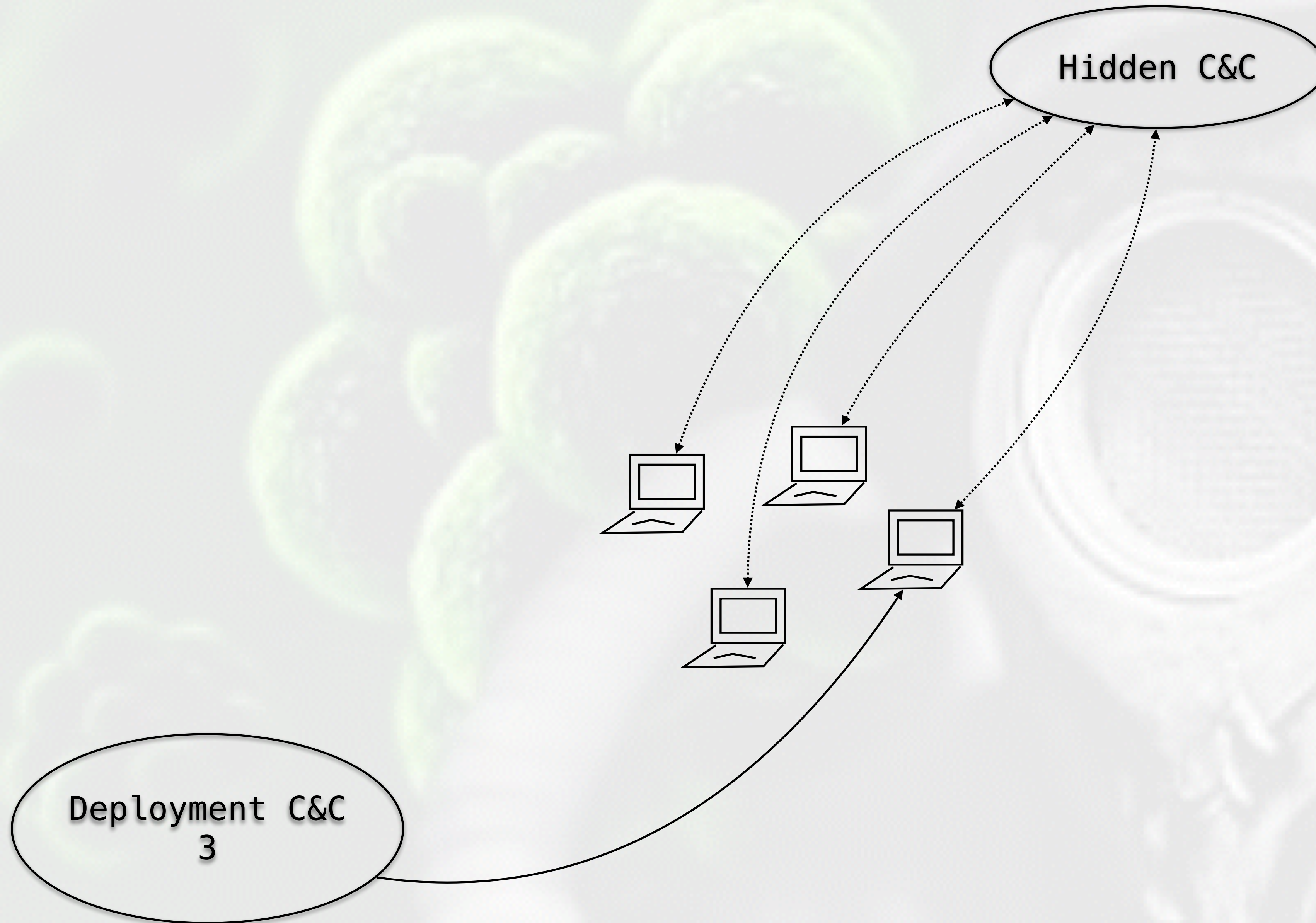


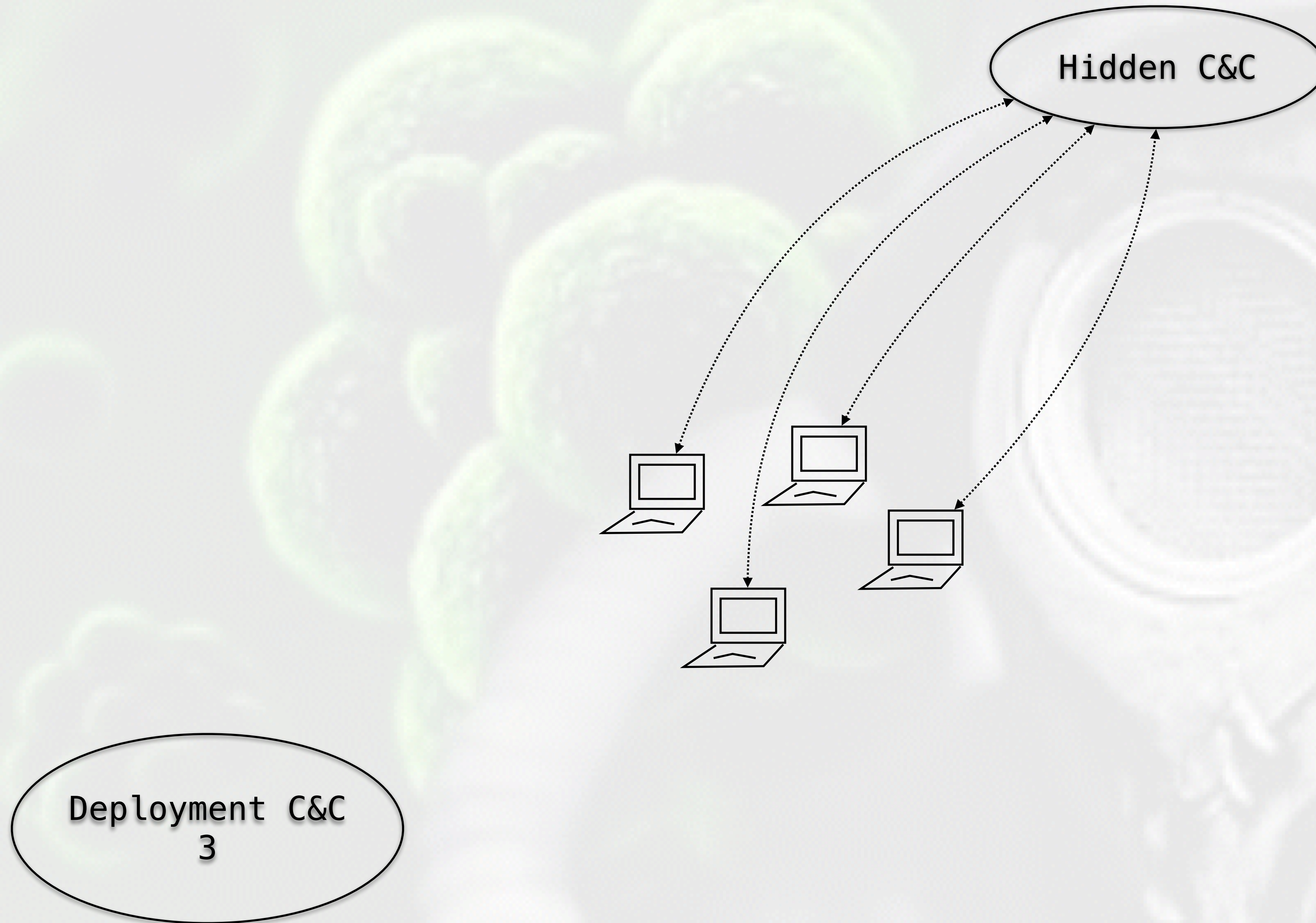


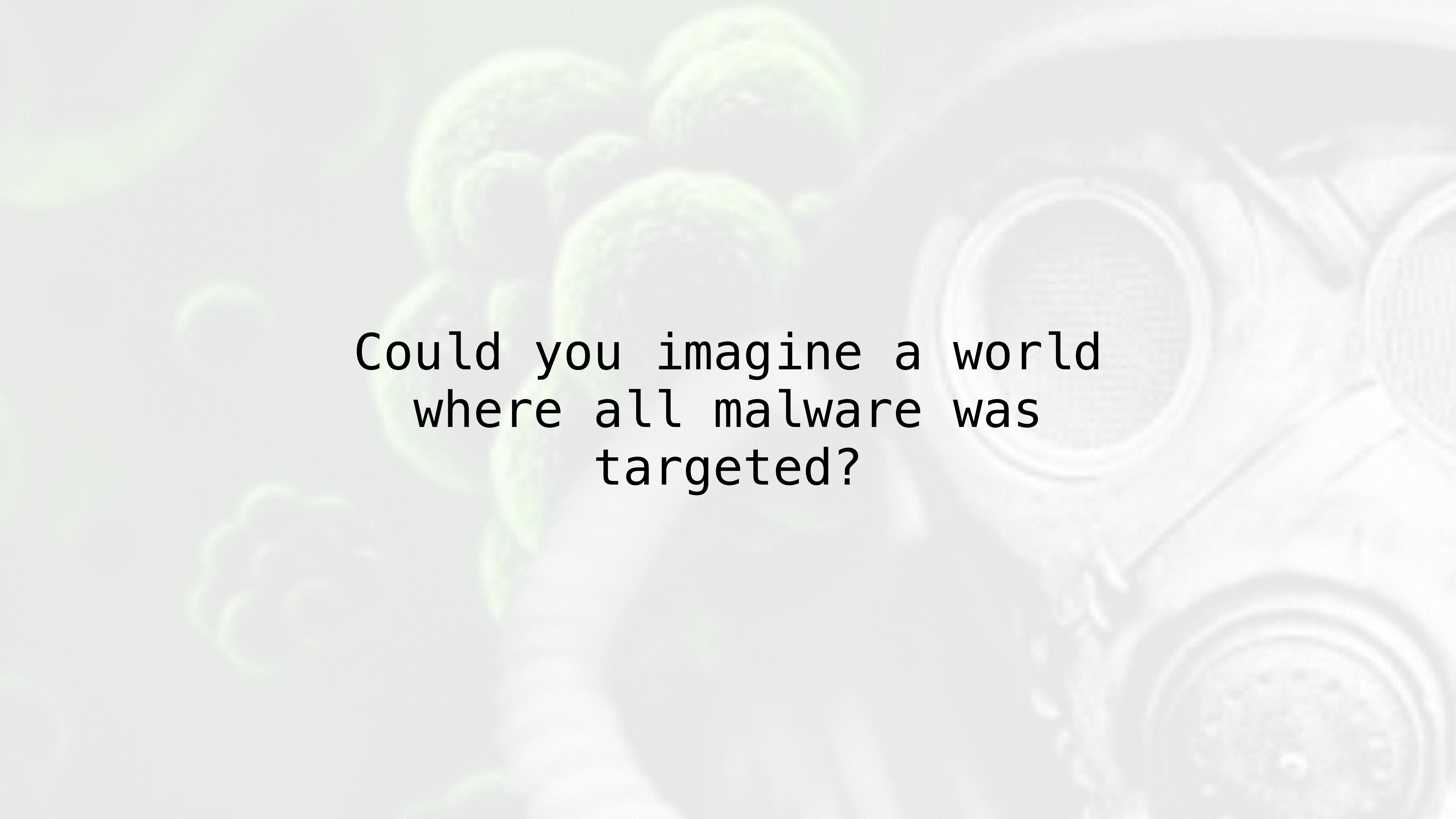




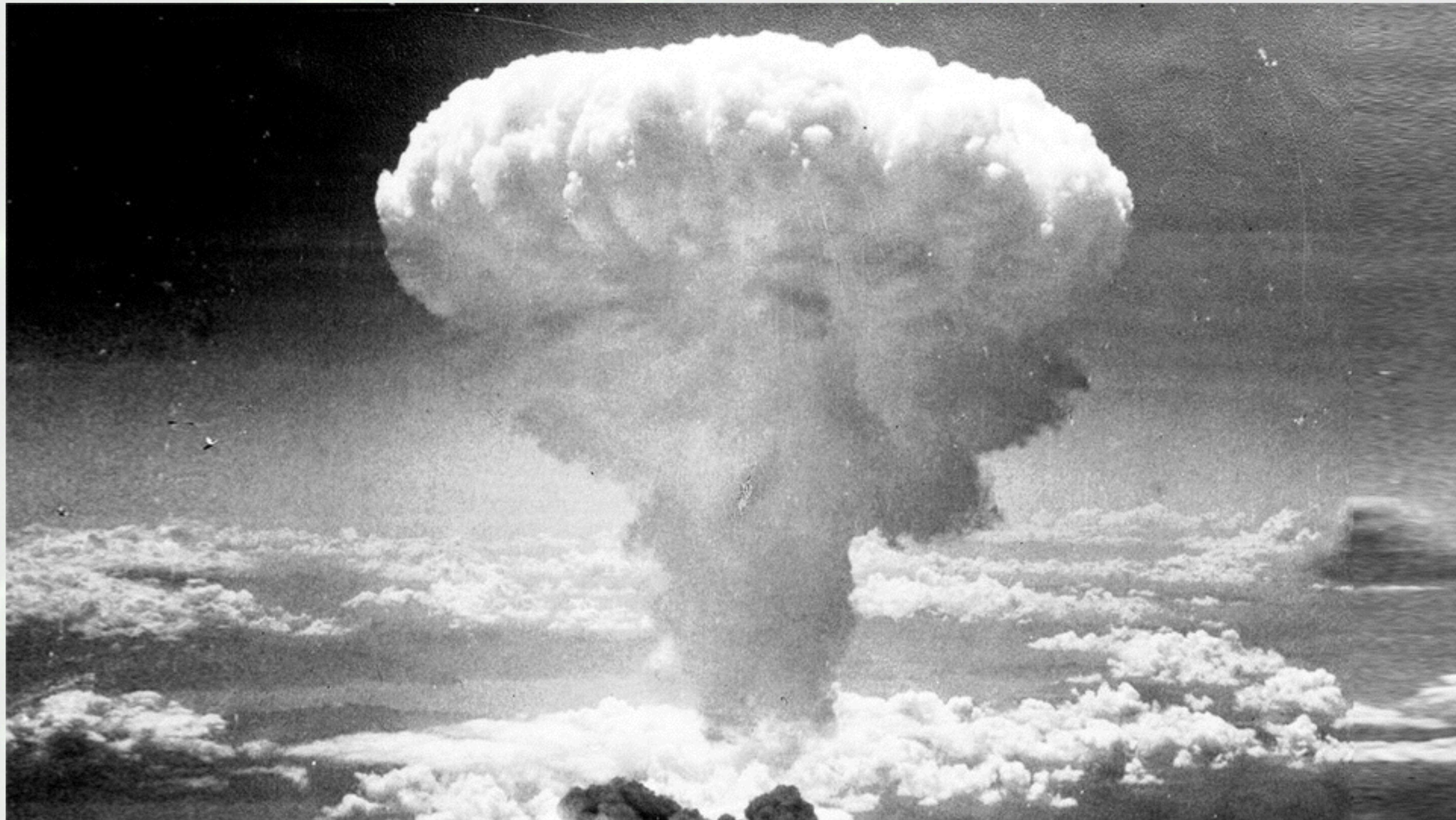








Could you imagine a world
where all malware was
targeted?









E.B.O.W.L.A.

E.B.O.W.L.A.

Ethnic BiO Weapon Limited Access

High Level Overview

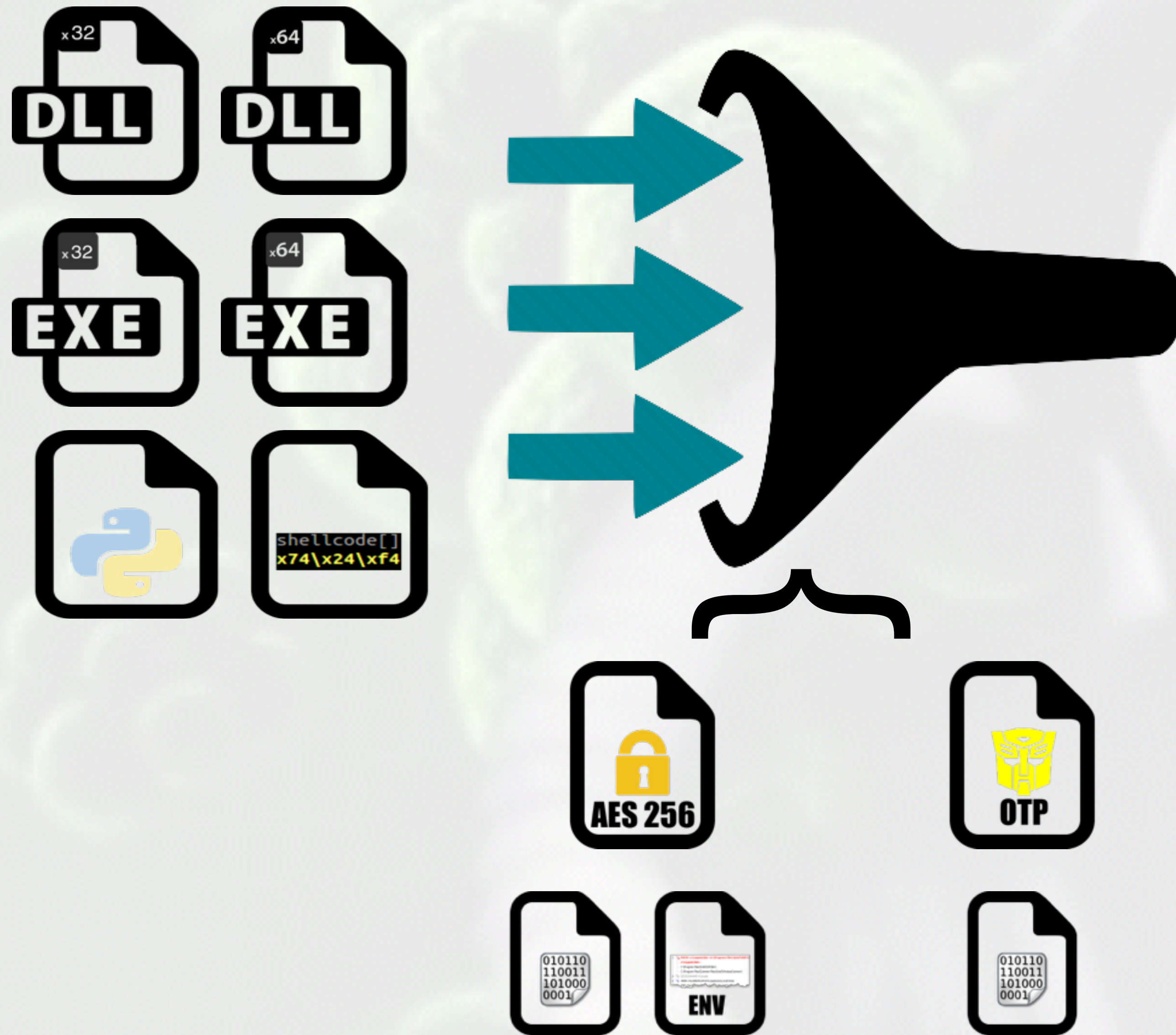




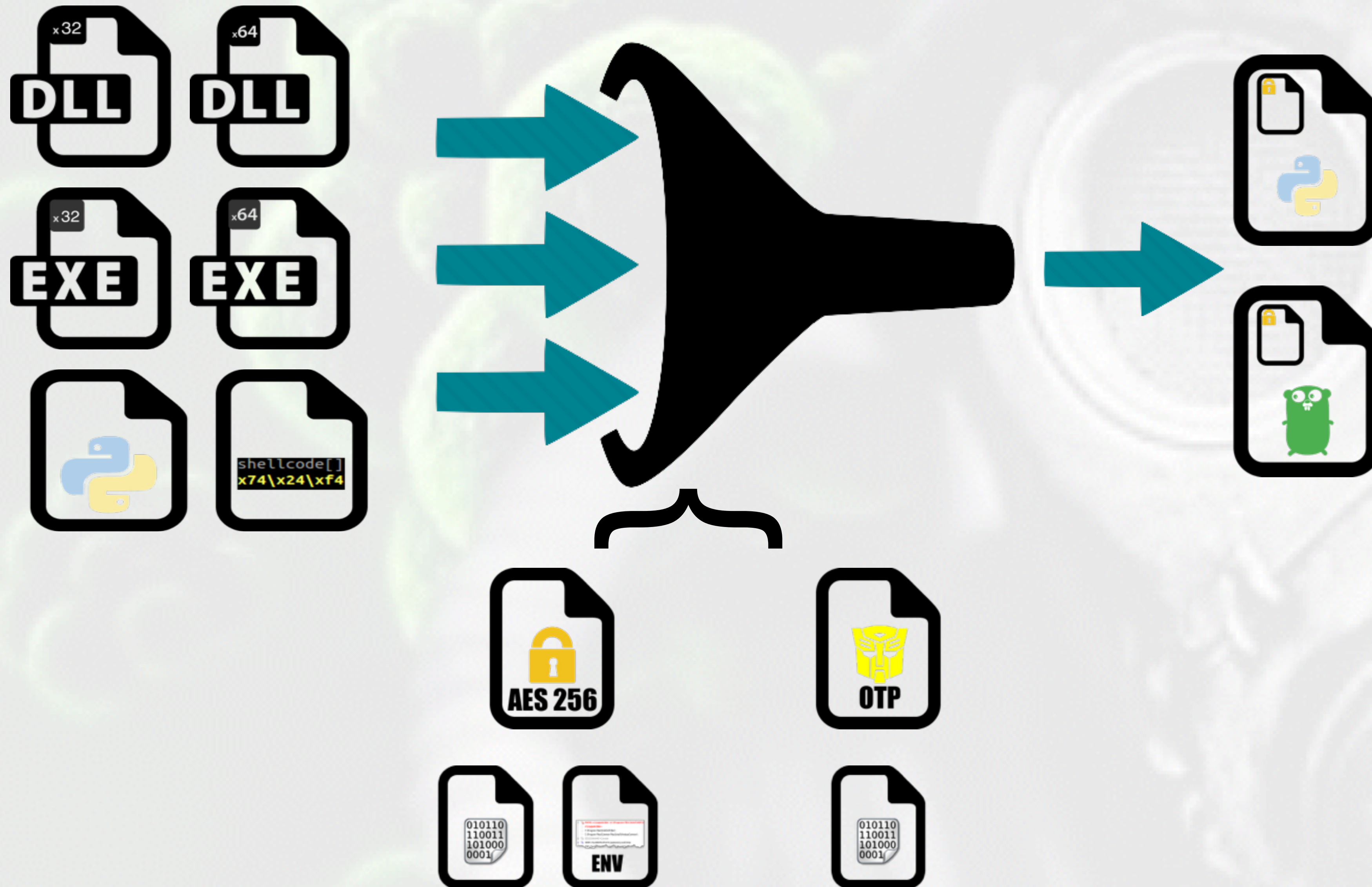
E.B.O.W.L.A.













EXPLOIT.W.L.A.













EB.O.W.L.A.




Framework


- ▼  genetic-malware
 - ▶  encryption
 - ▶  templates
 -  .gitignore
 -  __init__.py
 -  documentation.md
 -  ebowla.py
 -  genetic.config
 -  README.md
 -  roadmap.md


Framework


- ▼  genetic-malware
 - ▶  encryption
 - ▶  templates
 -  .gitignore
 -  __init__.py
 -  documentation.md
 -  ebowla.py
 -  genetic.config
 -  README.md
 -  roadmap.md


Framework


▼  genetic-malware

▼  encryption


 __init__.py


 env.py


 otp_full.py


 otp_key.py


Framework


▼  genetic-malware

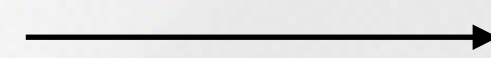
▼  encryption


 __init__.py

 env.py


 otp_full.py


 otp_key.py



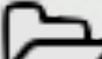






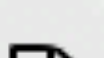
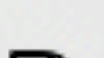
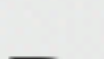
▼  templates

▶  go

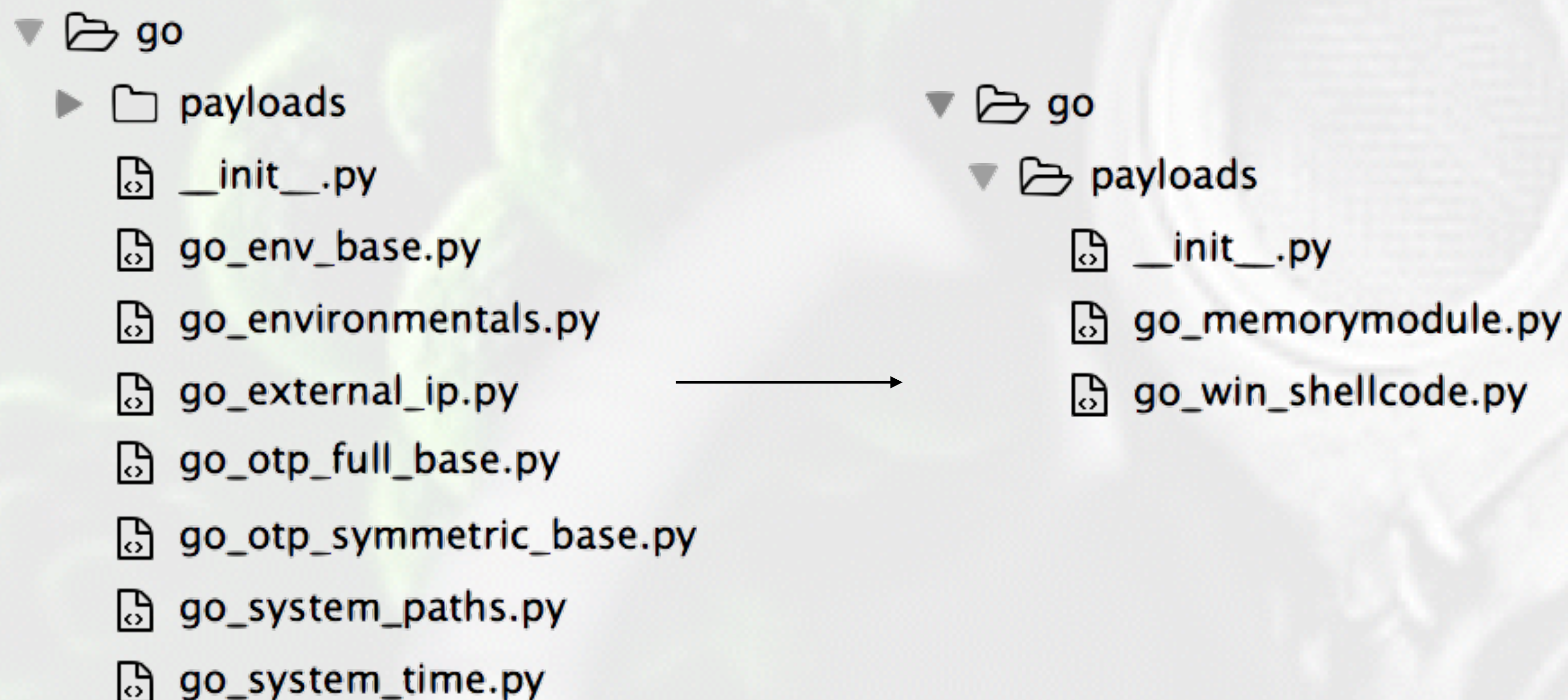
▶  python

 __init__.py

Framework

- ▼  go
 - ▶  payloads
 -  `__init__.py`
 -  `go_env_base.py`
 -  `go_environmentals.py`
 -  `go_external_ip.py`
 -  `go_otp_full_base.py`
 -  `go_otp_symmetric_base.py`
 -  `go_system_paths.py`
 -  `go_system_time.py`

Framework



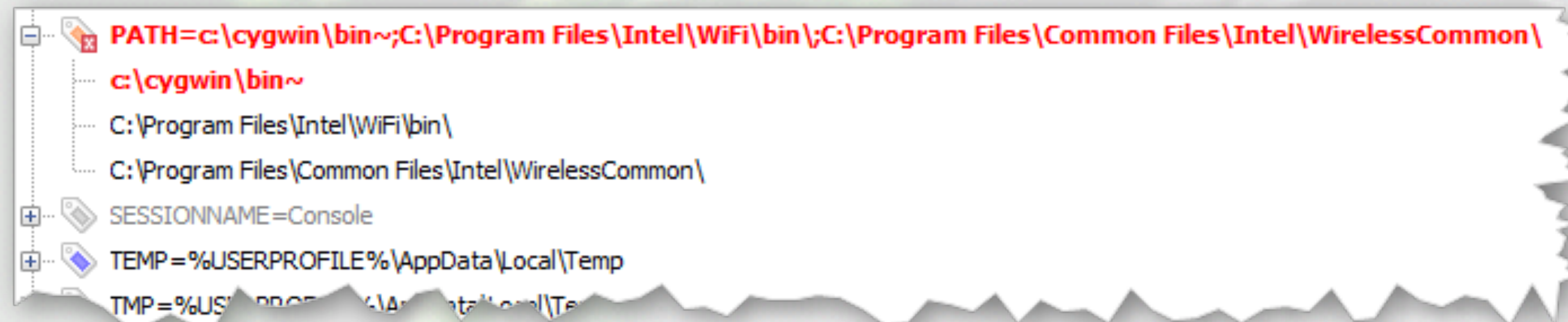
Protection Mechanisms



Protection Mechanisms



Key Derivation: Environmental Factors



Supported Environmentals

- Environment Variables (e.g. %TEMP%, %USERNAME%, %TEMP%, etc)
- File System Path (e.g. C:\windows\temp)
- External IP Range (e.g. 100.10.0.0, 100.0.0.0)
- Time Trigger (e.g. 20160401)

Key Derivation: Environmental Factors

Encryption:

```
payload_hash = sha512(payload[:-offset_bytes])  
key = ((sha512(token1+token2+...)) * Iterations)[:32]  
enc_blob = base64(zlib(iv+AES.CFB(key,iv,payload)))
```


Key Derivation: Environmental Factors

Encryption:

```
payload_hash = sha512(payload[:-offset_bytes])  
key = ((sha512(token1+token2+...)) * Iterations)[:32]  
enc_blob = base64(zlib(iv+AES.CFB(key,iv,payload)))
```

Decryption:

```
1) Retrieve environment variables  
2) Traverse File System from StartingPoint  
3) Combine into all possible combinations and decrypt  
  
** trial_key = sha512(token1 + token2 + ...)* Iterations)[:32]  
  
** if(sha512(decryptpayload(iv,enc_blob,trial_key[:-offset_bytes])) ==  
payload_hash; continue
```


Key Derivation: Unique File



Key Derivation: Unique File

Encryption:

```
payload_hash = sha512(payload[:-offset_bytes])  
location = rand_location(uniq_key_file)  
key = ((sha512(read.location) * Iterations)[:32]  
enc_blob = base64(zlib(location + lc.length + iv +  
AES.CFB(key,iv,payload)))
```


Key Derivation: Unique File

Encryption:

```
payload_hash = sha512(payload[:-offset_bytes])  
  
location = rand_location(uniq_key_file)  
  
key = ((sha512(read.location) * Iterations)[:32]  
  
enc_blob = base64(zlib(location + lc.length + iv +  
AES.CFB(key,iv,payload)))
```

Decryption:

- 1) Traverse File System from StartingPoint
- 2) Create a key from every file encountered & Attempt Decryption

```
** trial_key = sha512(readFile.location)* Iterations)[:32]  
  
** if(sha512(decryptpayload(iv,enc_blob[22:],trial_key)[:  
offset_bytes]) == payload_hash; continue
```


Protection Mechanisms



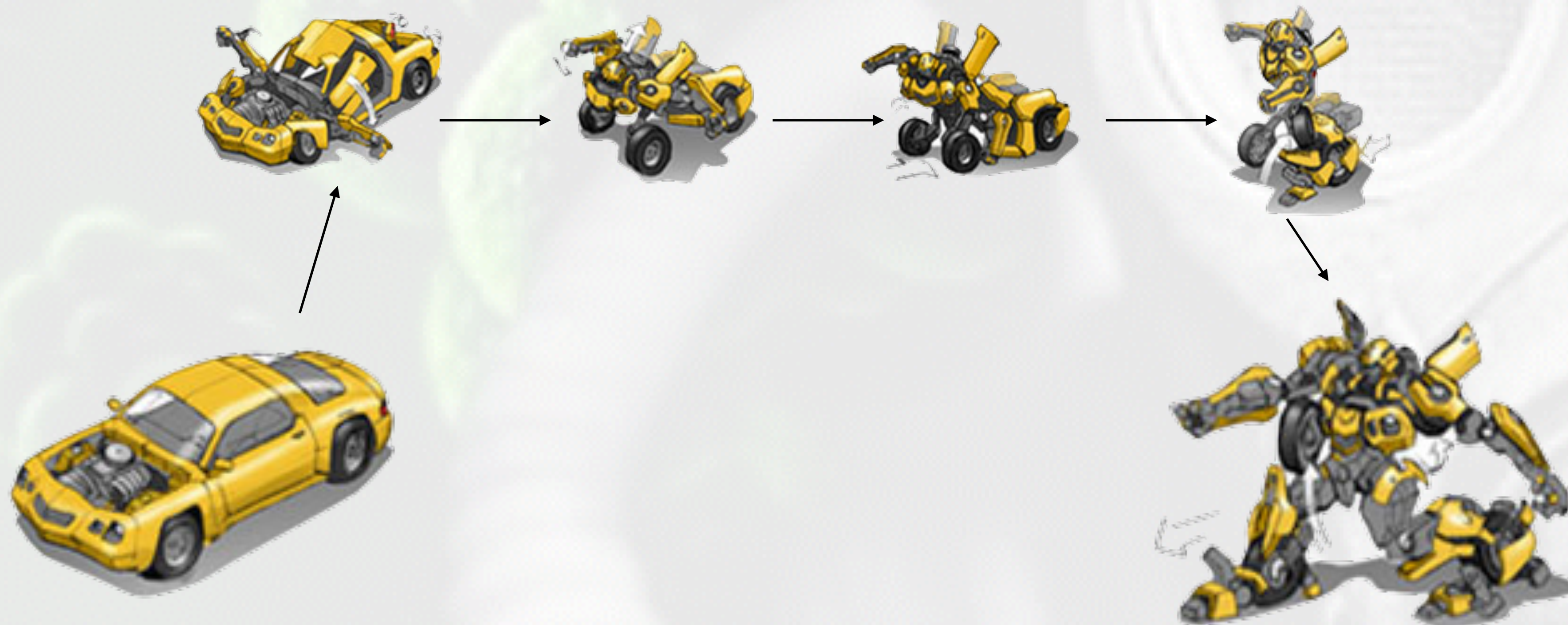
Protection Mechanisms



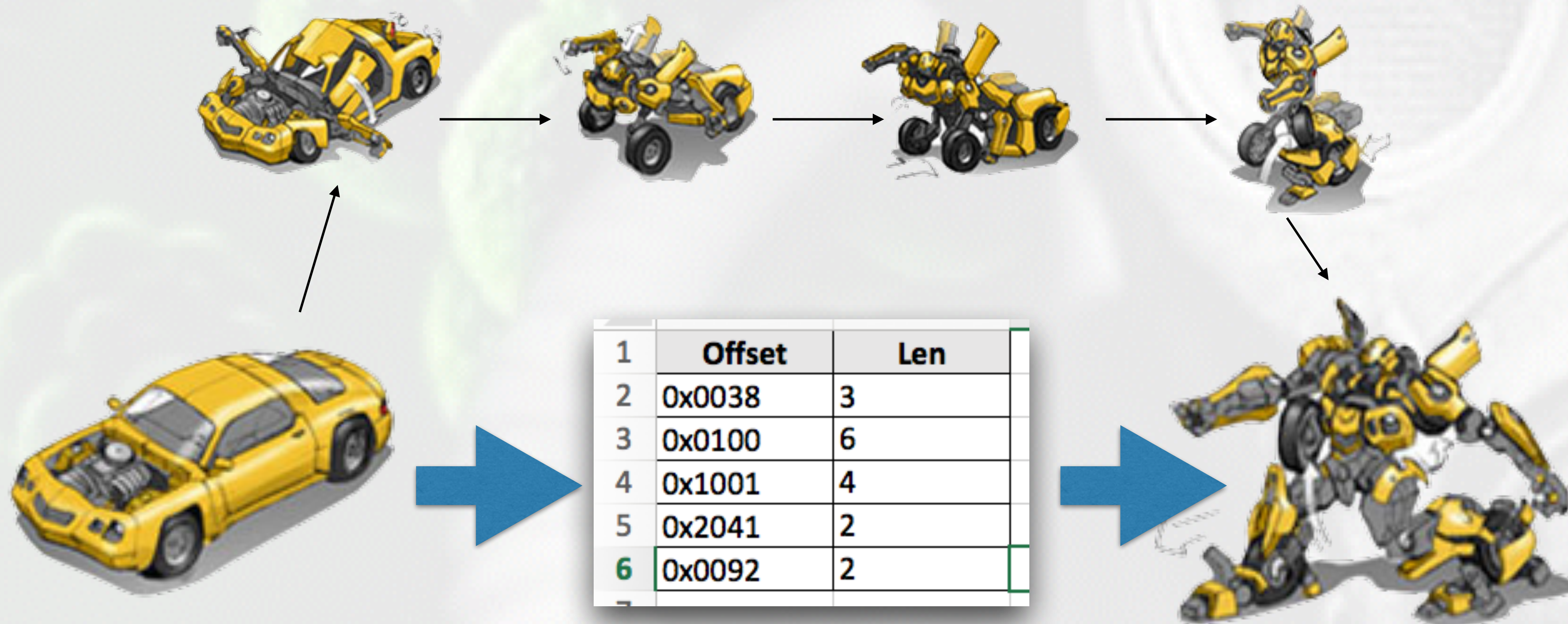
Protection Mechanisms



Key Derivation: One Time Pad (OTP)



Key Derivation: One Time Pad (OTP)



Key Derivation: One Time Pad (OTP)

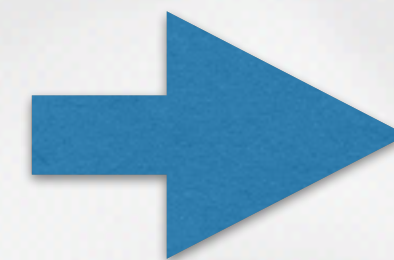
Pad Creation:

- 1) `payload_hash = sha512(payload[:-offset_bytes])`
- 2) `short_len = len(payload)*10%`
- 3) `payload_hash_short = sha512(payload)[:short_len]`
- 4) `lookup_table(uniqueBinary) = base64(zlib([[offset_loc][len], [offset_loc][len], ...]))`

Key Derivation: One Time Pad (OTP)

Offset	0	1	2	3	4	5	6	7	8
00000...	80	C0	00	20	DD	BE	00	00	CD
00000...	F5	AD	00	00	F9	AD	00	00	FD
00000...	05	AE	00	00	09	AE	00	00	0D
00000...	11	AE	00	00	15	AE	00	00	E5
00000...	19	AE	00	00	1D	AE	00	00	21
00000...	29	AE	00	00	2D	AE	00	00	31
00000...	39	AE	00	00	3D	AE	00	00	41

Attacker Payload



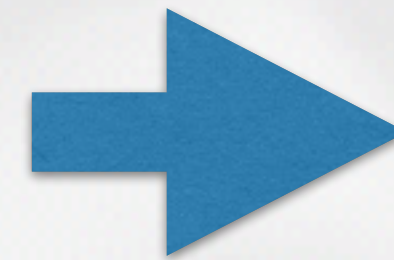
DD	BE	00	00	CD	AD	00	00	F1
F9	AD	00	00	FD	AD	00	00	01
09	AE	00	00	0D	AE	00	00	D9
15	AE	00	00	E5	AD	00	00	55
1D	AE	00	00	21	AE	00	00	25
2D	AE	00	00	31	AE	00	00	35
3D	AE	00	00	41	AE	00	00	45
3D	AE	00	00	41	AE	00	00	45

Target UniqueBinary

Key Derivation: One Time Pad (OTP)

Offset	0	1	2	3	4	5	6	7	8
00000...	80	C0	00	20	DD	BE	00	00	CD
00000...	F5	AD	00	00	F9	AD	00	00	FD
00000...	05	AE	00	00	09	AE	00	00	0D
00000...	11	AE	00	00	15	AE	00	00	E5
00000...	19	AE	00	00	1D	AE	00	00	21
00000...	29	AE	00	00	2D	AE	00	00	31
00000...	39	AE	00	00	3D	AE	00	00	41

Attacker Payload



DD	BE	00	00	CD	AD	00	00	F1
F9	AD	00	00	FD	AD	00	00	01
09	AE	00	00	0D	AE	00	00	D9
15	AE	00	00	E5	AD	00	00	55
1D	AE	00	00	21	AE	00	00	25
2D	AE	00	00	31	AE	00	00	35
3D	AE	00	00	41	AE	00	00	45
4D	AE	00	00	51	AE	00	00	55
5D	AE	00	00	61	AE	00	00	65

Target UniqueBinary

Lookup Table

1	Offset	Len
2	0x0038	3
3		
4		

Key Derivation: One Time Pad (OTP)

Decryption:

- 1) Traverse File System from StartingPoint
- 2) Open Each file and build 10%
- 3) Validate 10% hash Matches then build entire payload

```
** if(sha512(rebuild_payload(lookup_table,current_file)[:  
offset_bytes] == payload_hash; exec()
```


Outputs

(aka Cyber Pathogens)



Warning: Dormant
Cyber Pathogen

Outputs



GO



Python



Input/Out Compatibility

Payload	Python		GO	
	x64	x32	x64	x32
Reflective DLL			In Memory	In Memory
DLL			In Memory	In Memory
EXE	On Disk	On Disk	In Memory	In Memory
ShellCode	In Memory	In Memory	In Memory	In Memory
Python Code	In Memory	In Memory		



Usage

The background of the slide features a faded, light-colored image of a car engine on the right side, showing various components like the air filter and belts. On the left side, there are green, leafy plants, possibly ferns, which are also faded. The overall background is a light gray.

`$./ebowla.py payload config`

`$ #Then compile output`



The config file

Three Sections

- Overall
- OTP Settings
- Symmetric Settings

Overall Section

Encryption_Type
OPTIONS: OTP ENV

output_type
OPTIONS: Python, GO, Both

payload_type
OPTIONS for GO: EXE, DLL_x86, DLL_x64, SHELLCODE
OPTIONS for PYTHON: EXE, SHELLCODE, CODE

key_iterations
OPTIONS: Any number? Be reasonable.

OTP Settings

otp_type

OPTIONS: full, key

pad

Any file you want. Make sure it has 0–256 bytes represented.

pad_max

Maximum size your pad, support up $256 \times 3 - 1$ (~16MB)

scan_dir

start location for finding the pad

OPTIONS: A fixed path OR an environment variable such as %APPDATA%

byte_width

For use with OTP FULL only

Nominal for speed 8–12

The larger the number the longer it takes to build on the attacker's side, but faster to rebuild on the client side.

OPTIONS: A Single number, Example: 8

Symmetric Key Settings

This has four sections:

- ENV_VARS
- PATH
- IP_RANGES
- SYSTEM_TIME

Symmetric Key Settings

ENV_VARS

Can be anything, can add whatever you want
if value is '', it is not used. The value is used as a key.

examples:

username = 'Administrator' # Used

homepath = '' # Not used

PATH

path

This is used as a key.

OPTIONS: A full static path.

start_loc

Location to start looking for path match

OPTIONS: Static location or Env variable (%PROGRAMFILES%)

Symmetric Key Settings

IP_RANGES

external_ip_mask

Simple IP MASK, limited to /24 /16 /8

Example: 11.12.13.14, 11.12.13.0, 11.12.0.0, 11.0.0.0

SYSTEM_TIME

Time_Range

Limited to Year, Month, or DAY

Format: YYYYMMDD

Example: 20160401, 20160400, or 20160000

DEMO TIME



DEMO TIME



The Scenario

- An American in Moscow is low on Rubles
- Wants Starcraft really bad
- Answer: BitTorrent a cracked game!
- Unfortunately the cracked starcraft games are patched with a backdoor targeting the most current version of BitTorrent

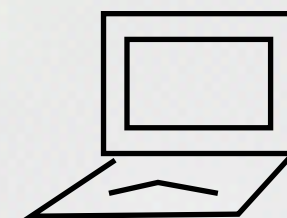
DEMO 1: OTP

- Using BitTorrent.exe as the PAD
 - Version 7.9.5, Build 41866, 32bit
 - Meterpreter reverse https is the payload via a first stage DLL
 - Searching for the PAD starts in %APPDATA%
- Code delivered through a backdoored/cracked game
 - Download and Execute payload

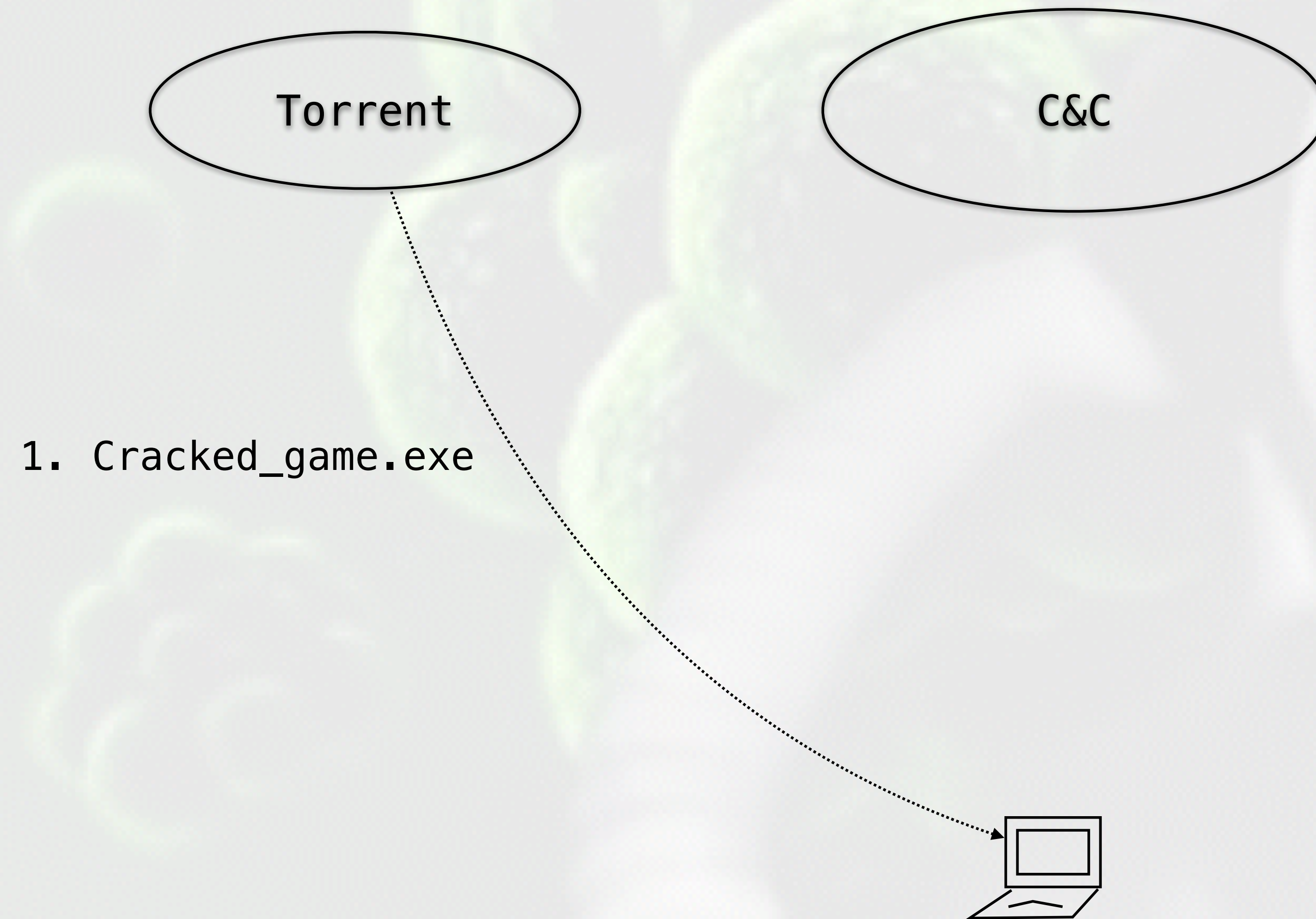
DEMO 1: OTP

Torrent

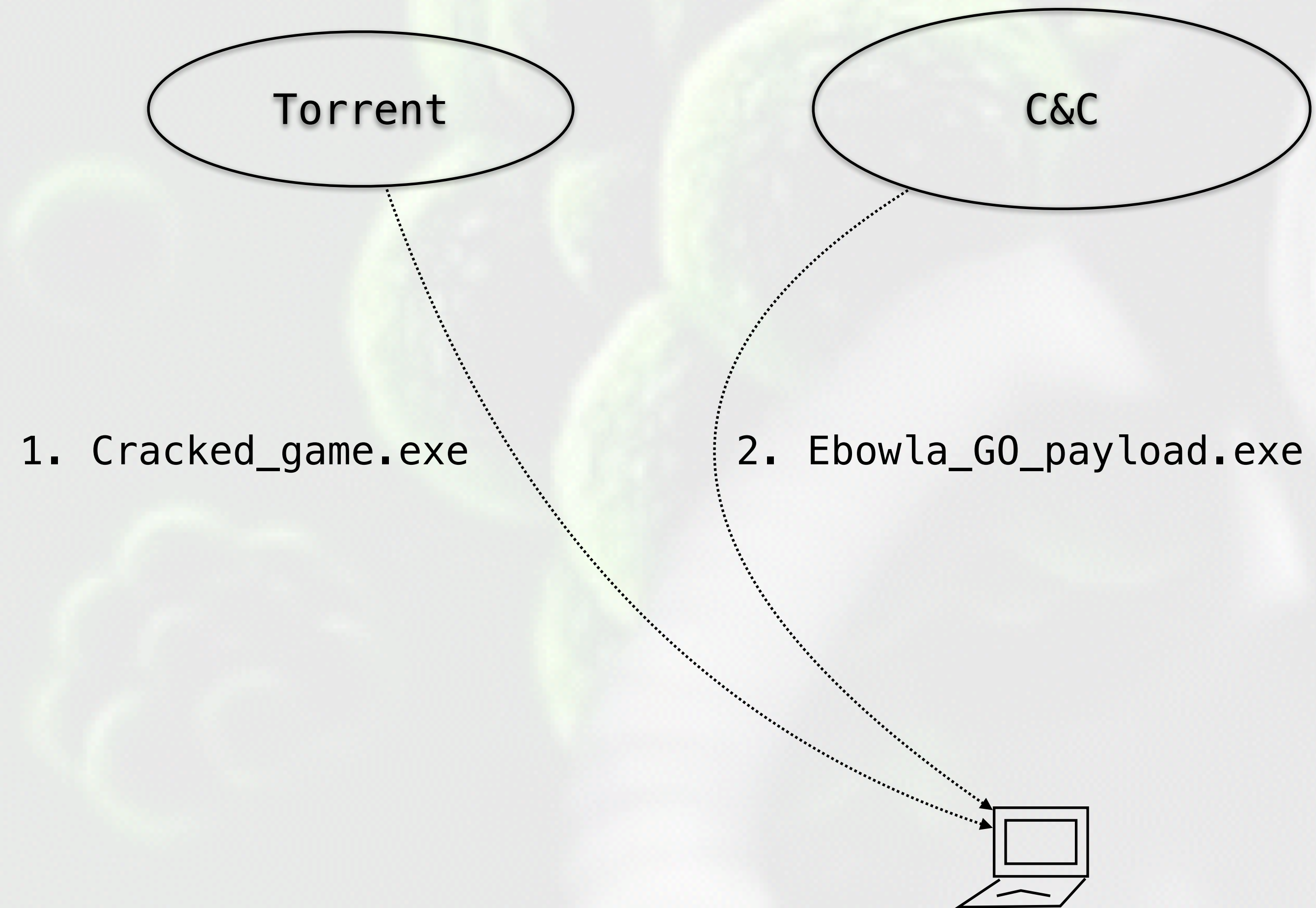
C&C



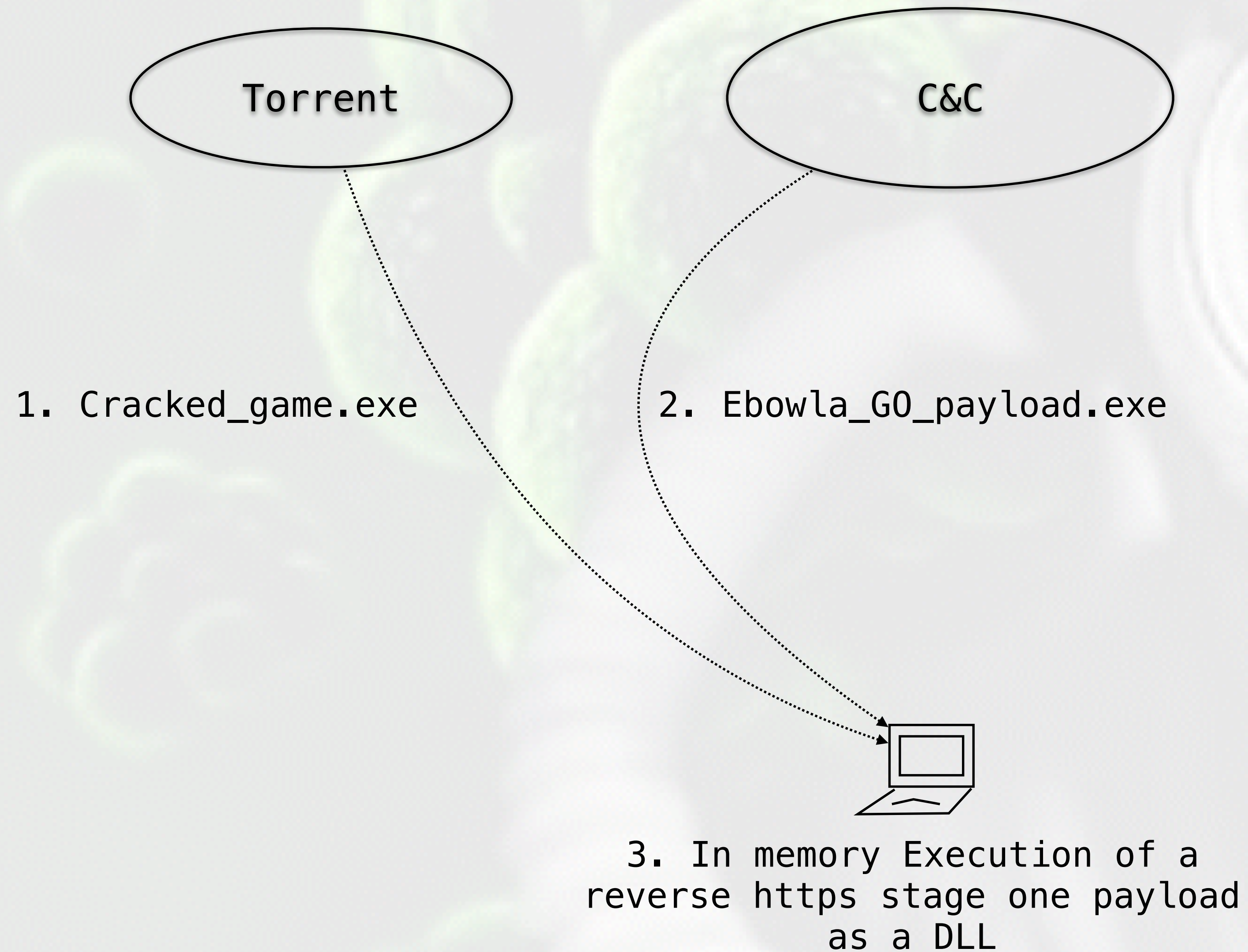
DEMO 1: OTP



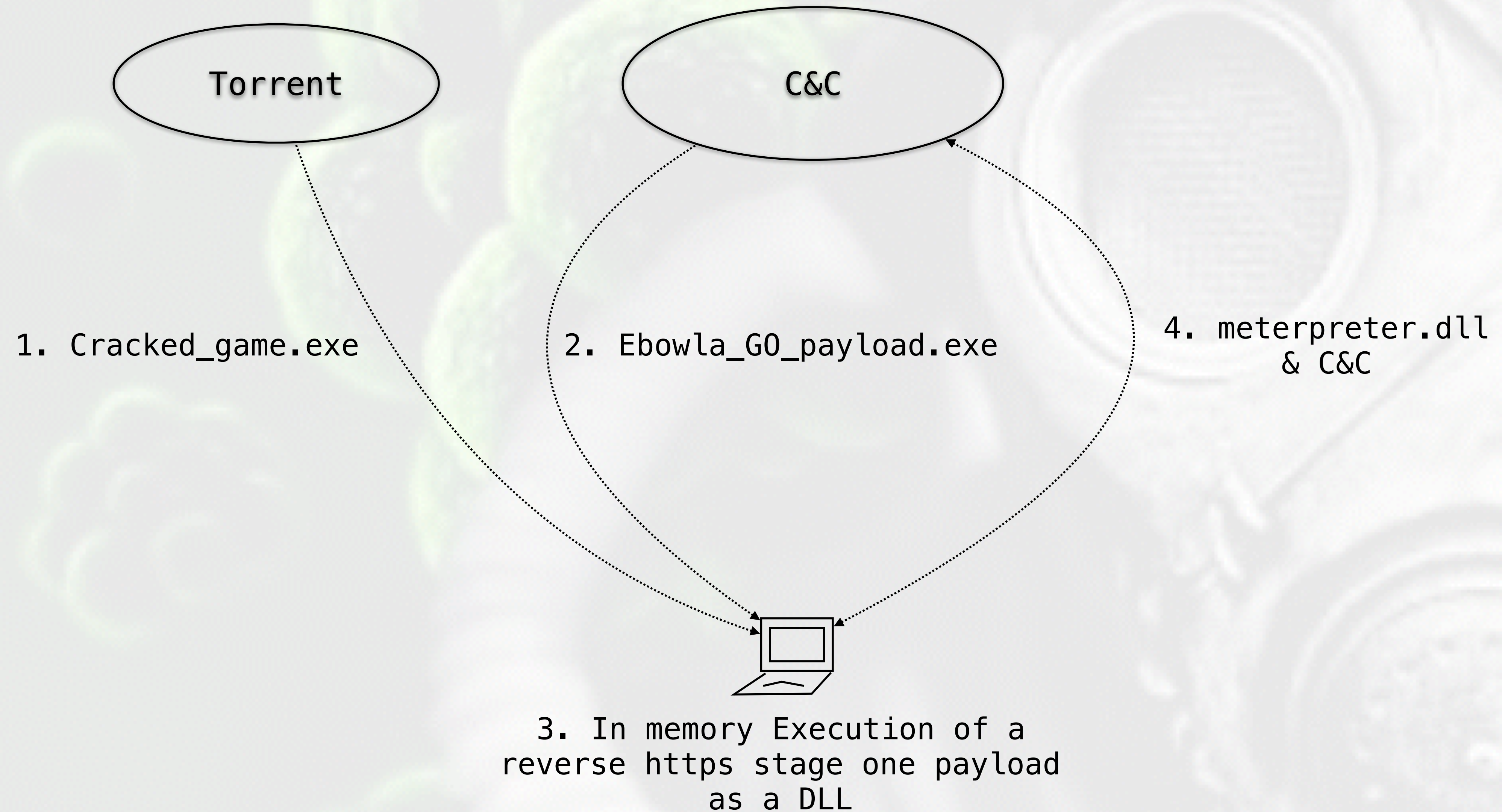
DEMO 1: OTP



DEMO 1: OTP



DEMO 1: OTP



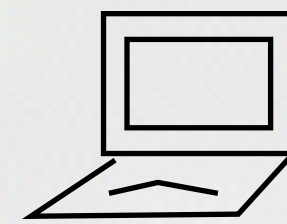
DEMO 2: Key from File

- Using a location in BitTorrent.exe as the AES key source
- Version 7.9.5, Build 41866, 32bit
- Pupy EXE reverse https
- Searching starts in %APPDATA%
- Code delivered through a backdoored/cracked game
- Download and Execute payload

DEMO 2: Key from File

Torrent

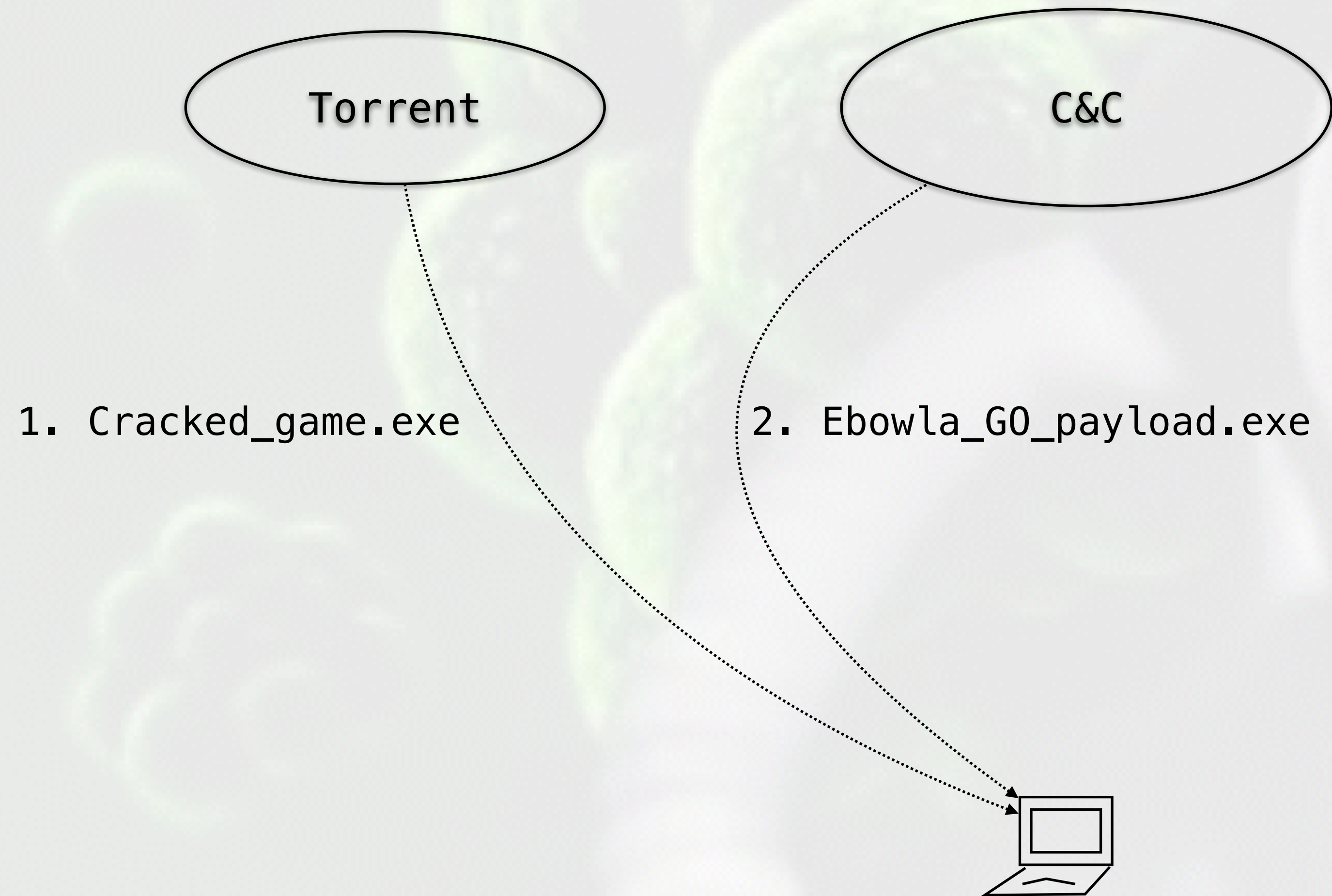
C&C



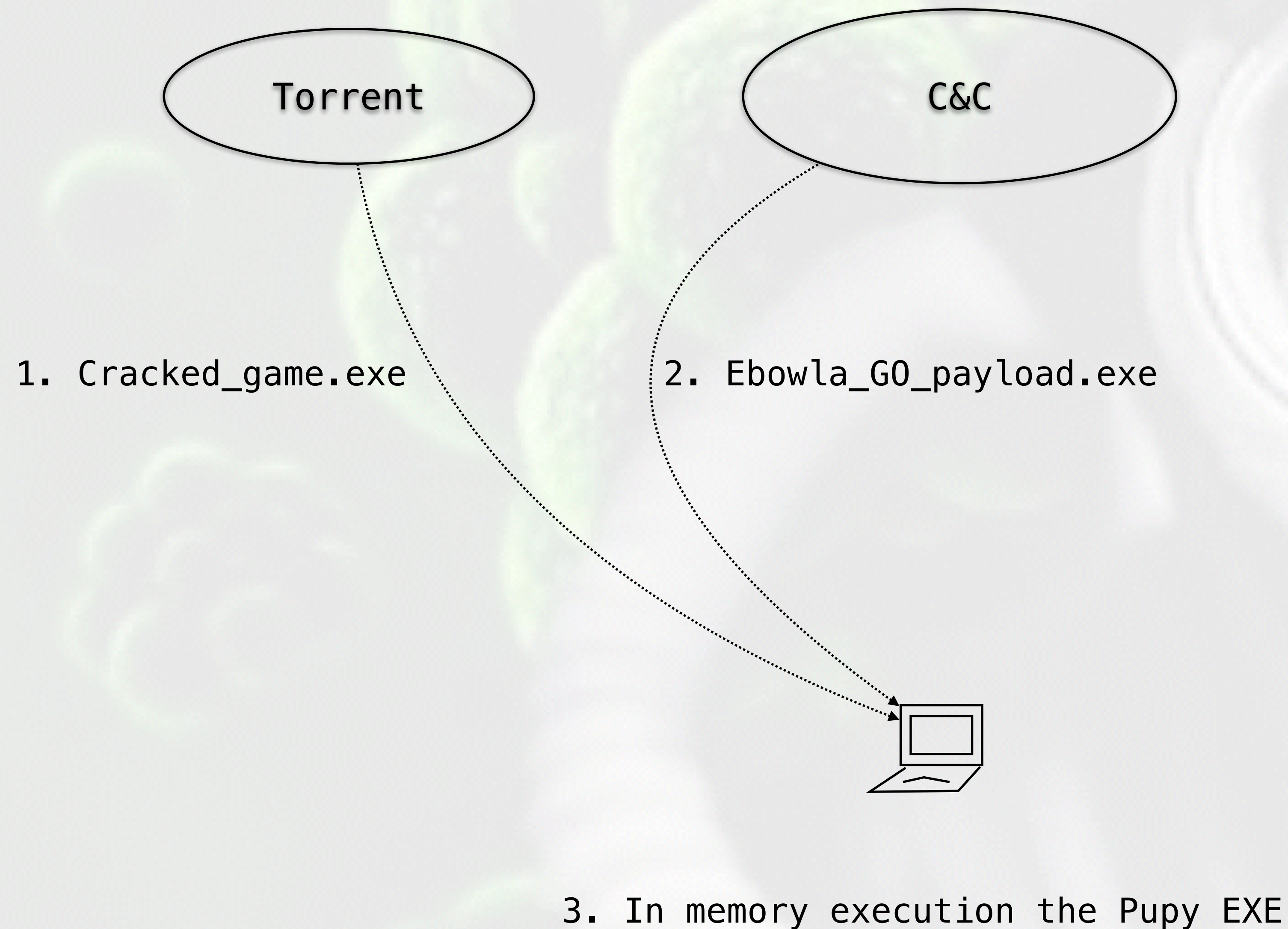
DEMO 2: Key from File



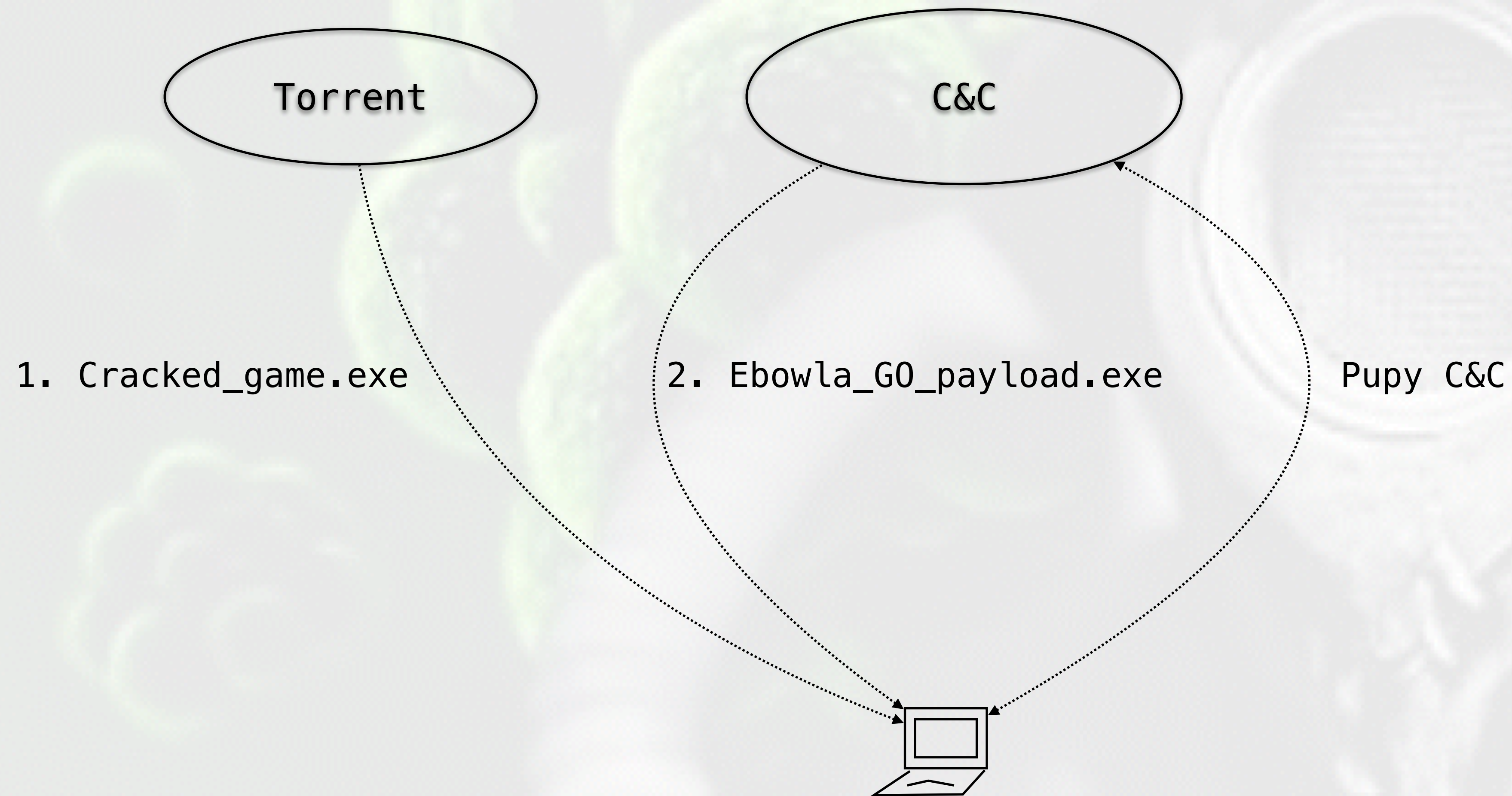
DEMO 2: Key from File



DEMO 2: Key from File



DEMO 2: Key from File



3. In memory execution the Pupy EXE

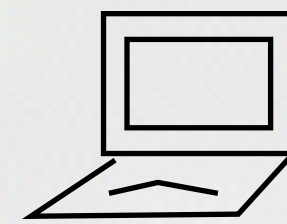
DEMO 3: Layered Payload

- Using Environmental Factors
- Stage 2:
 - Env Vars: Computer Name, number of processors as keys
 - GO EXE launching Pupy x64 DLL
- Stage 1:
 - Using Date Range and IP Mask as keys
 - Python EXE, writes stage 1 to disk and Executes
- Code delivered through a backdoored/cracked game
 - Download and Execute payload

DEMO 3: Layered Payload

Torrent

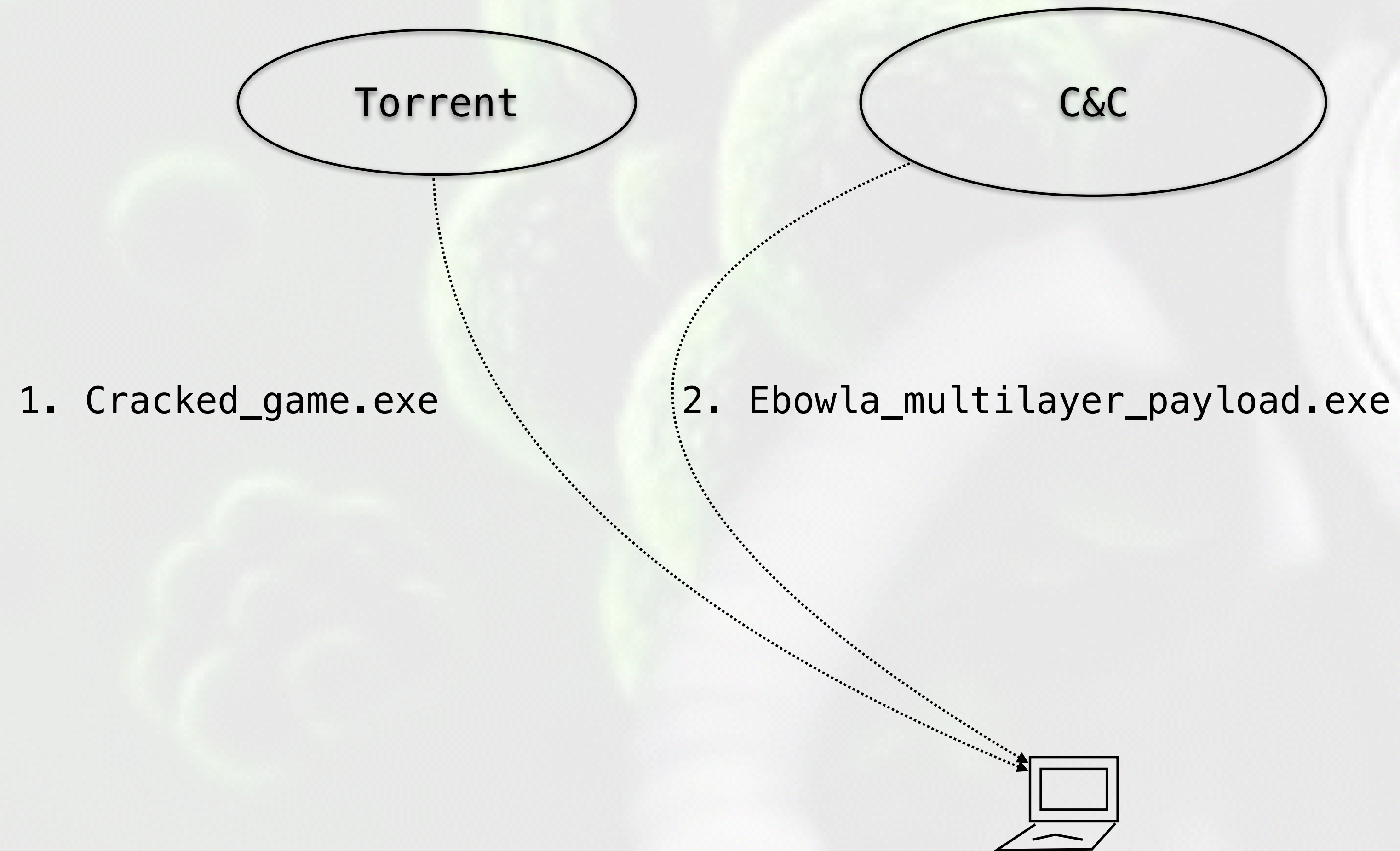
C&C



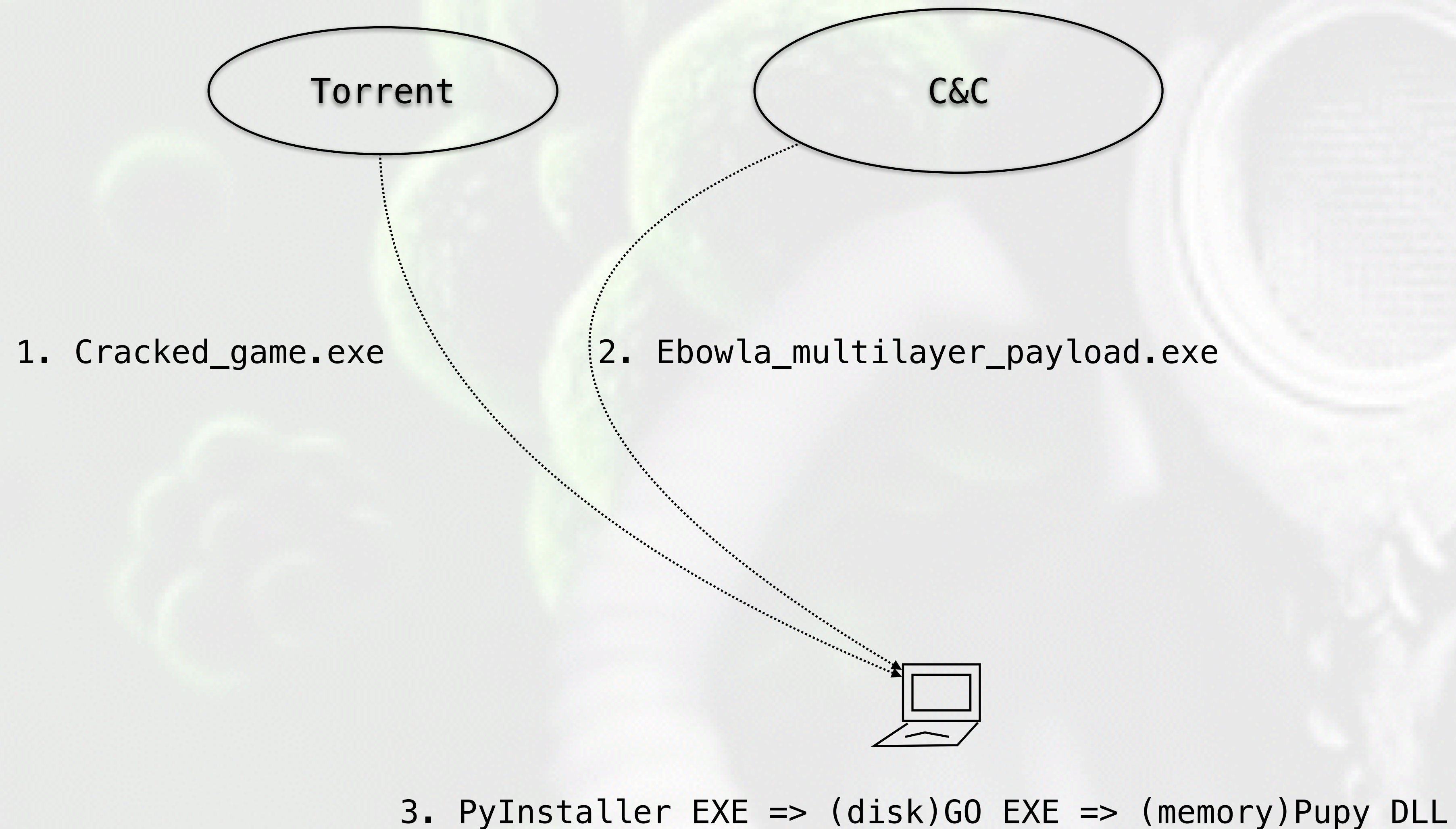
DEMO 3: Layered Payload



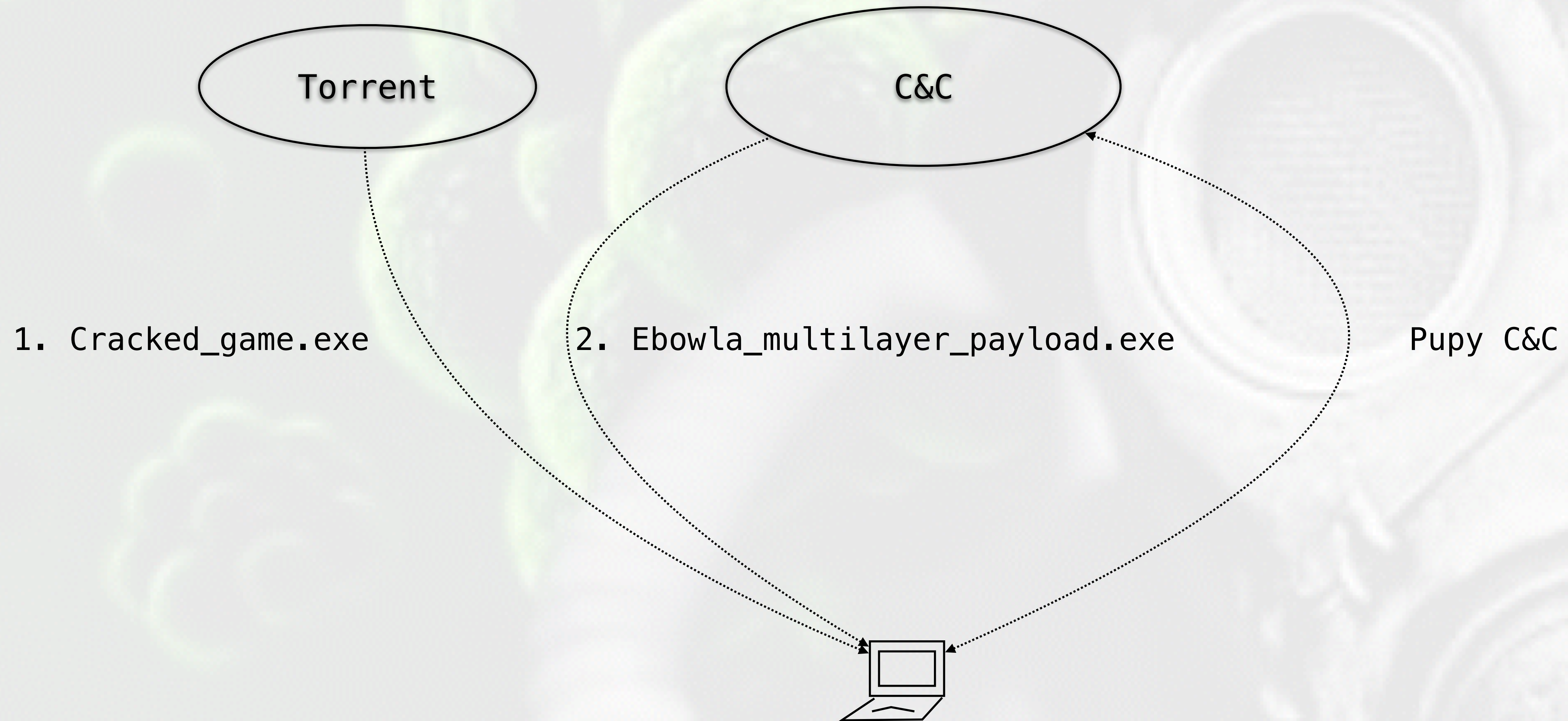
DEMO 3: Layered Payload



DEMO 3: Layered Payload



DEMO 3: Layered Payload



3. PyInstaller EXE => (disk)GO EXE => (memory)Pupy DLL

Known Issues/Bugs

- Previous knowledge requirement
- Chaining payloads:
 - GO EXE launching GO via Memory Module – DIE IN A FIRE
 - Pyinstaller EXE launching Pyinstaller EXE FROM DISK – Loses namespace
 - GO (memory module) -> Pyinstaller – Just no...
 - Metasploit x86 PE EXE template does not work with MemoryModule
- OTP:
 - MZ/DOS Header Leak

This is OK

- Go EXE
- PyInstaller EXE
- Chaining PyInstaller EXE -> GO EXE

Roadmap

- C/C++ loaders/output
- Reflective DLL
- Better chaining of payloads
- OSX/NIX Support

Questions?

Download: <https://www.github.com/genetic-malware/Ebola>

@wired33

@midnite_runr

Credits

<https://github.com/vyrus001/go-mimikatz>

https://archive.org/details/P-G_0hst_Exploitation

<https://matrixbob.files.wordpress.com/2015/03/bio-weapons.gif>

<http://blogs-images.forbes.com/benkerschberg/files/2015/02/crowdsourcing-spigot.jpg>

<http://static5.businessinsider.com/image/51e418a66bb3f7230a00000e-1200-900/guys-drinking-coffee-in-tel-aviv.jpg>