Samsara Metamorphic Engine

Michael Grube
6/5/22

About Me

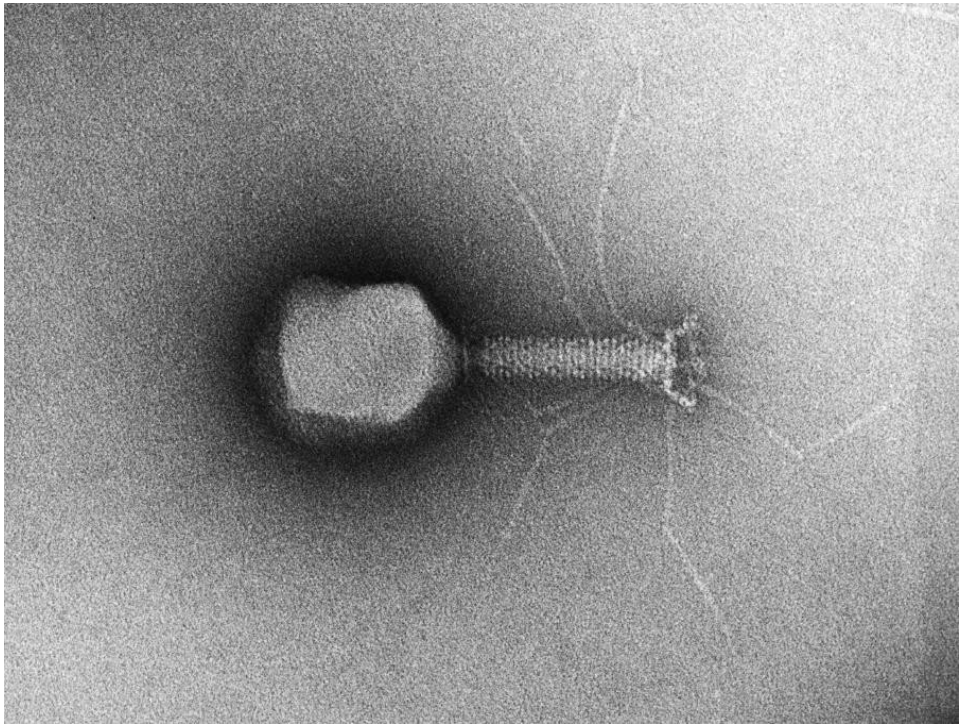
- Former Software Developer
- Taught Malware Classes
- First sec job in 2016
- Just quit SpaceX

Motivations

- Spend more time in userland
- Study viruses
- Study compilers
- Study the JVM

Virus

- **Malware != Virus**
- **Oxford Dictionary:**
  - A piece of code which is capable of copying itself and typically has a detrimental effect
- **My definition:**
  - Any program that can inject a full copy of itself into a target program without damaging the target's original code flow

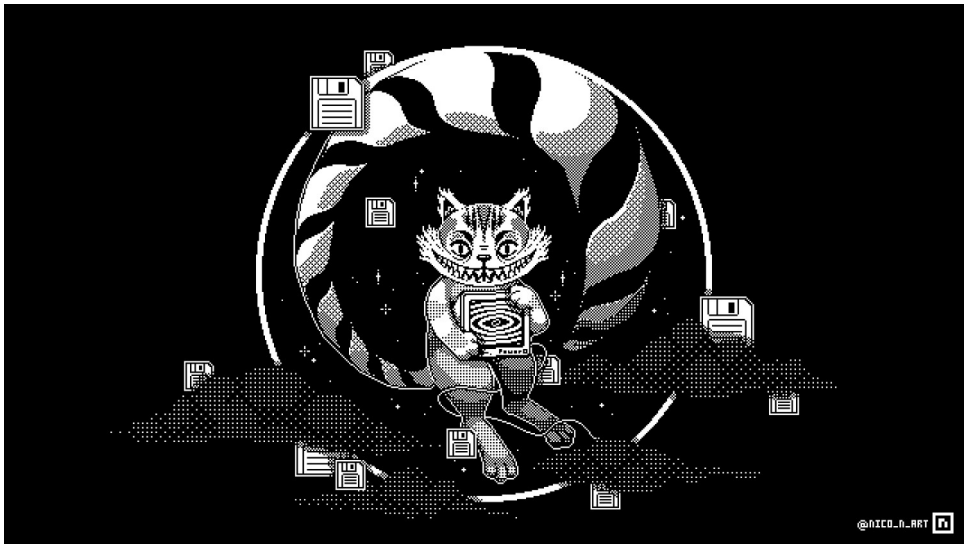


Advantages

- Can spread on the filesystem, in memory, over the network
- Whole class of executables can now be your malware
- Ideally, code is small - KBs
- Self-propagation makes attribution more difficult

Cheshire

- JVM is a good target for this case
  - WASM
  - Python Wheels
  - Others
- First publicly documented JVM virus in 20 years
- Strangebrew is the only other virus
- https://bit.ly/38PeTme



?

Community Score

9 security vendors and no sandboxes flagged this file as malicious

a632a1e7891b267449661331a3537a4fe345f7bc744cdb6a7d5c5a87fcb117d1

virus.jar

jar

18.11 KB
Size

2022-02-19 23:59:27 UTC
2 days ago

JAR

DETECTION | DETAILS | RELATIONS | BEHAVIOR | COMMUNITY

Ad-Aware	Virus.Java.Cheshire.1	Arcabit	Virus.Java.Cheshire.1
BitDefender	Virus.Java.Cheshire.1	Emsisoft	Virus.Java.Cheshire.1 (B)
eScan	Virus.Java.Cheshire.1	ESET-NOD32	A Variant Of Java/Cheshire.A
GData	Virus.Java.Cheshire.1	MAX	Malware (ai Score=83)
Trellix (FireEye)	Virus.Java.Cheshire.1	AhnLab-V3	Undetected
Alibaba	Undetected	ALYac	Undetected

ID	10038732
Released	Jul 02, 2021
Description Updated	Jul 02, 2021

Detection Availability

FortiGate	✔
Extended	✔
FortiClient	✔
Extreme	✔
FortiAPS	✔
FortiAPU	✔
FortiMail	✔

Threat Encyclopedia

Java/Cheshire.A

Analysis

Java/Cheshire.A is classified as a file infector.
A file infector is a type of malware that has the capability to propagate by attaching its code to other programs or files.
The Fortinet Antivirus Analyst Team is constantly updating our descriptions. Please check the FortiGuard Encyclopedia regularly for updates.

Recommended Action

- Make sure that your FortiGate/FortiClient system is using the latest AV database.
- Quarantine/delete files that are detected and replace infected files with clean backup copies.

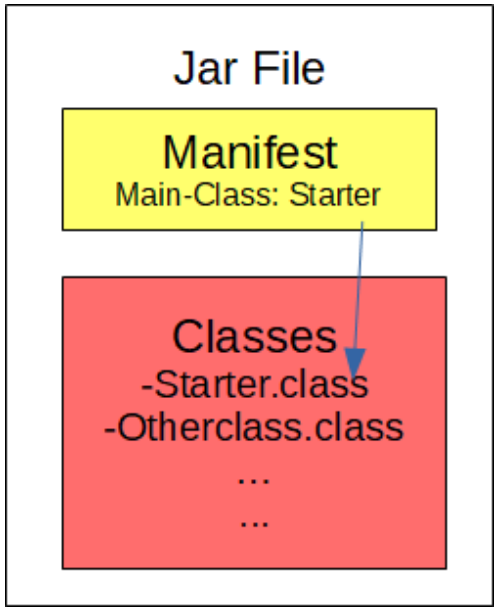
Telemetry

The Infection Process

- Load parser into memory
- Read yourself
- Read the target
- Add items to the constant pool
- Methods injected
- Output written to target

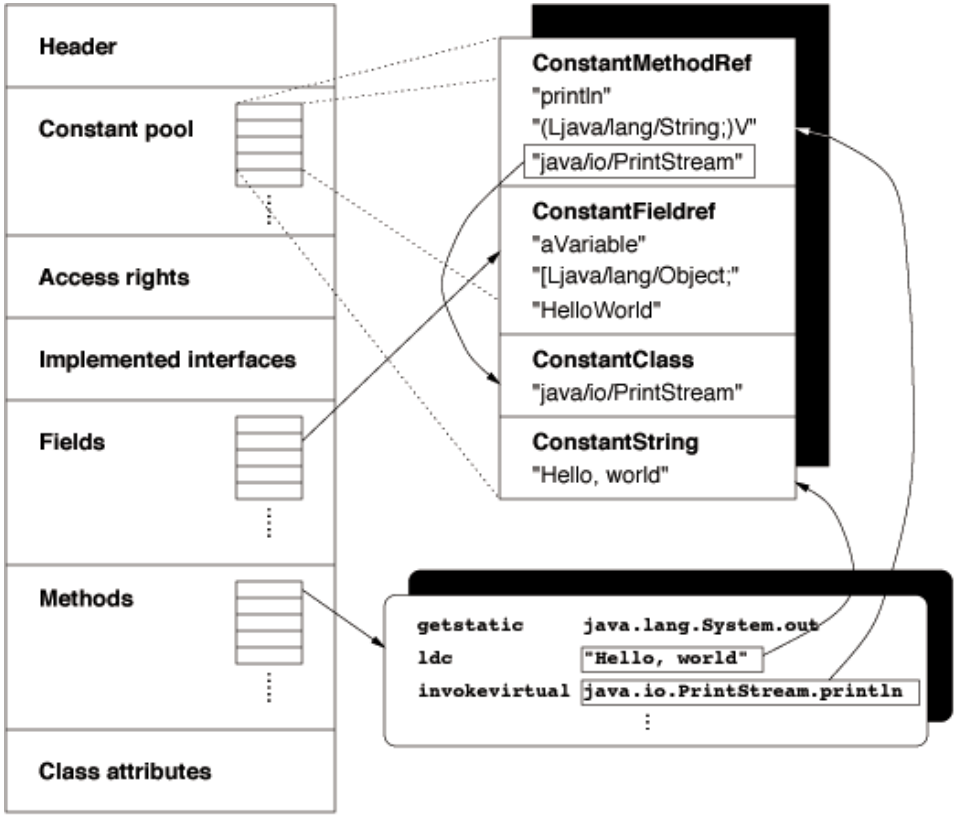
Entry point

- By default, jars execute the main method of the Main-Class specified in the manifest
- Any persistent place to begin execution works
- Inject an invokestatic into the main method



Viral Code Replication

- Need to reliably trigger execution with methods alone
  - Static methods don't require an object to be created
- Copy methods
- Adjust all instructions and references to the constant pool
- Adjust the stackmaptable as necessary
- Add calling instruction to main method of target



Samsara Metamorphic Virus

- Self-Modifying
- Easier to use
- Learn from the AV signatures that were created for Cheshire

Polymorphism and Metamorphism

- **Polymorphism**
  - Not your dad's polymorphism
  - Change any aspect of code not related to instructions
  - Examples:
    - Cryptographic Keys
    - Functions and Variable Names
    - Function and Variable Order
- **Metamorphism**
  - Instructions are loaded into memory and re-written
  - Examples:
    - Cryptographic algorithms
    - Code instructions
    - Function arguments

Metamorphic Engine Anatomy

- Disassembly
- Simplification
- Metamorphic Routines
- Recompilation

Disassembly

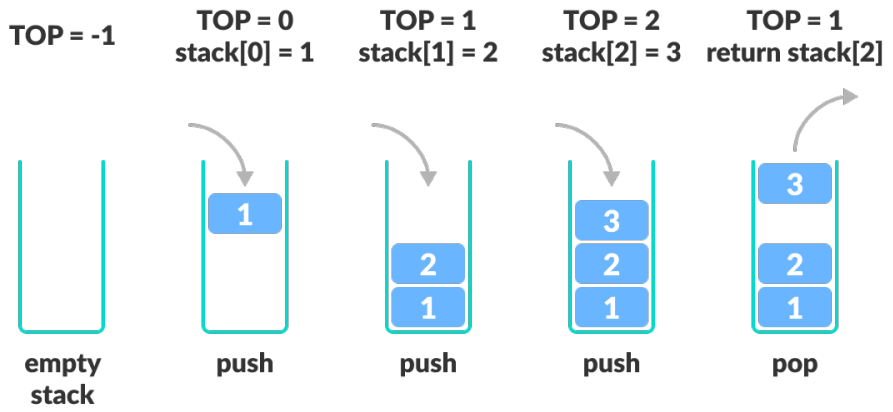
- < 255 instructions
- Variable length instructions
- Relatively simple

Metamorphic Techniques

- Stack addition/subtraction
- Junk instructions
- Indirect calls

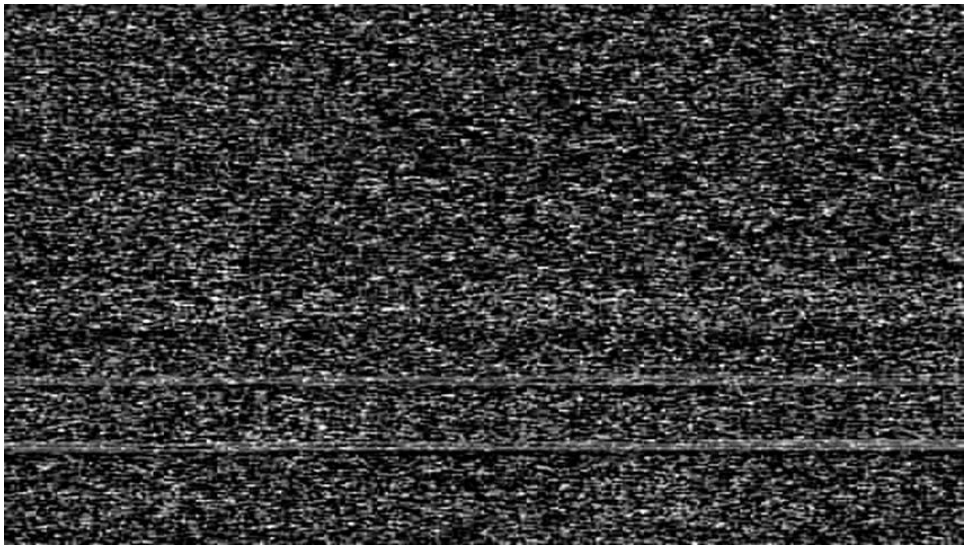
Stack Manipulation

- The JVM is a *stack machine*
  - This means return values are passed and returned entirely on the stack
  - We can disrupt static analysis by placing items on the stack so long as the correct items are in place at the time of calling



Junk Instructions

- Place random constant pool string on stack, do nothing with it
- Math operations
- Object creation



Indirect Calls

- Java Reflection!
- Replace invoke instructions with invokes of Method.invoke
- Invoke takes a string argument, which can be obfuscated

Code simplification

- **Detect reflective calls to our own methods, return them to normal invokes**
- **Find operations that do nothing**
- **Detect unnecessary stack items**
  - Walk back from invocations, emulate arguments until they match

Compilation

- Make sure instructions point to valid resources
- Ensure Type compatibility at every jump target

Java Bytecode Verification

- Type safety is a core feature of Java
- Local variable and stack types are validated to some extent at the instruction level
- More specific type enforcement happens with the help of a *Stack Map*
- Makes injecting arbitrary instructions *very annoying*

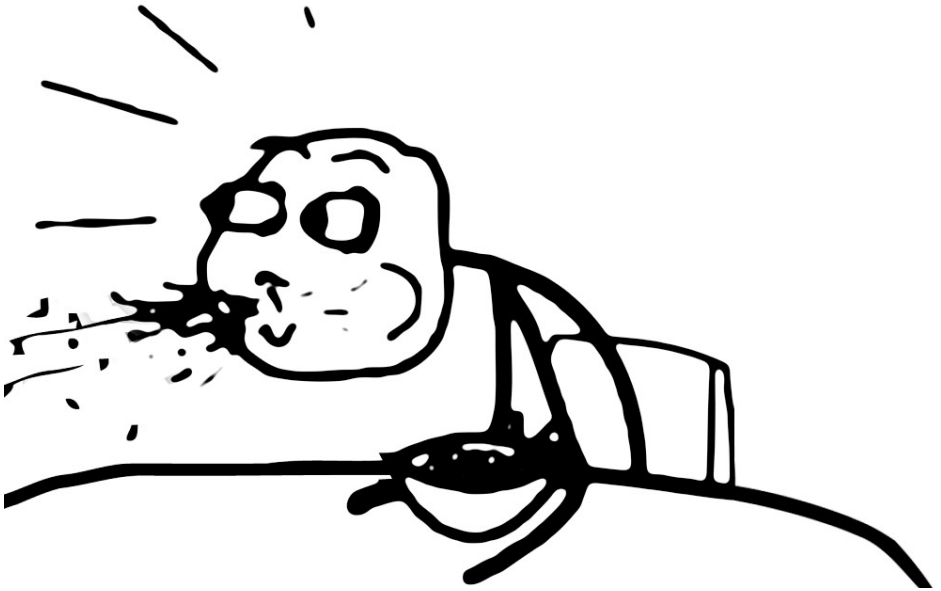
```
Bytecode instruction 47 is first occurrence of jump target 85
Bytecode instruction 1327 is first occurrence of jump target 1379
Bytecode instruction 1455 is first occurrence of jump target 1584
Bytecode instruction 1584 is first occurrence of jump target 1423
Bytecode instruction 1398 is first occurrence of jump target 24
Bytecode instruction 188 is first occurrence of jump target 267
Bytecode instruction 1597 is first occurrence of jump target 1618
Handler target 2144 reached.
Vars accessed in this frame:
[0, 4, 5, 6]
Frame vars:{0=[B, 1=Ljava/util/HashMap;, 2=Ljava/util/HashMap;, 3=Ljava/util/ArrayList;, 4=Ljava/nio/ByteBuffer;, 5=Ljava/io/ByteArrayOutputStream;, 6=Ljava/util/ArrayList;}
Max in keyset: 6
Number of framevars: 7
Framevar info: [7, 0, -121, 7, 0, 12, 7, 0, 12, 7, 0, 18, 7, 1, -79, 7, 0, -120, 7, 0, 18]
Stackvars: []
Vars accessed in this frame:
[0, 4, 5, 6, 7, 8, 9]
Frame vars:{0=[B, 1=Ljava/util/HashMap;, 2=Ljava/util/HashMap;, 3=Ljava/util/ArrayList;, 4=Ljava/nio/ByteBuffer;, 5=Ljava/io/ByteArrayOutputStream;, 6=Ljava/util/ArrayList;, 7=B, 8=B, 9=[B}
Max in keyset: 9
Number of framevars: 10
Framevar info: [7, 0, -121, 7, 0, 12, 7, 0, 12, 7, 0, 18, 7, 1, -79, 7, 0, -120, 7, 0, 18, 1, 1, 7, 0, -121]
Stackvars: []
Vars accessed in this frame:
[0, 4, 5, 6, 7, 8, 9]
Frame vars:{0=[B, 1=Ljava/util/HashMap;, 2=Ljava/util/HashMap;, 3=Ljava/util/ArrayList;, 4=Ljava/nio/ByteBuffer;, 5=Ljava/io/ByteArrayOutputStream;, 6=Ljava/util/ArrayList;, 7=B, 8=B, 9=[B}
Max in keyset: 9
Number of framevars: 10
Framevar info: [7, 0, -121, 7, 0, 12, 7, 0, 12, 7, 0, 18, 7, 1, -79, 7, 0, -120, 7, 0, 18, 1, 1, 7, 0, -121]
Stackvars: []
```

Anti-Analysis

- Attribute Stripping
- Hiding User Payload

Hiding the User Payload

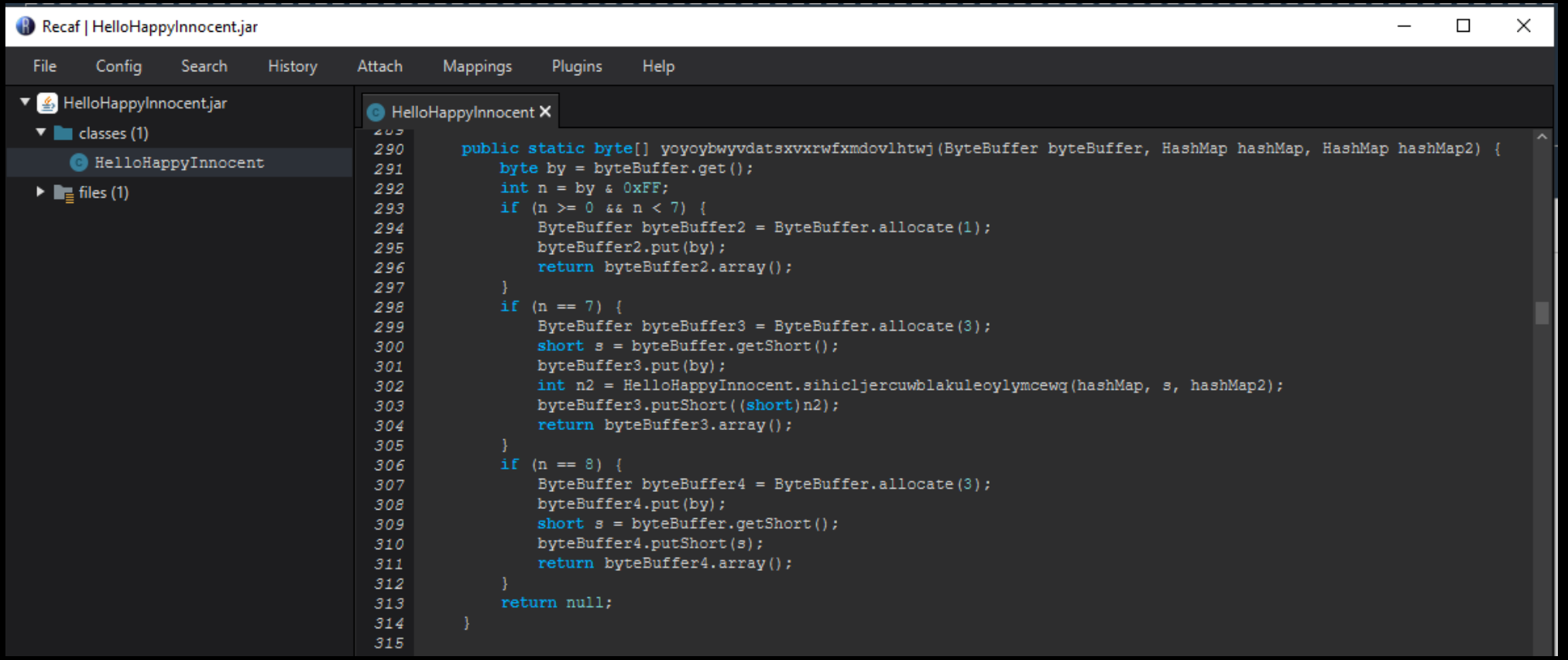
- **Reading the Java Specification:**
  - "Java Virtual Machine implementations are required to silently ignore attributes they do not recognize."
  - Java's accepted max attribute size is 65535 bytes

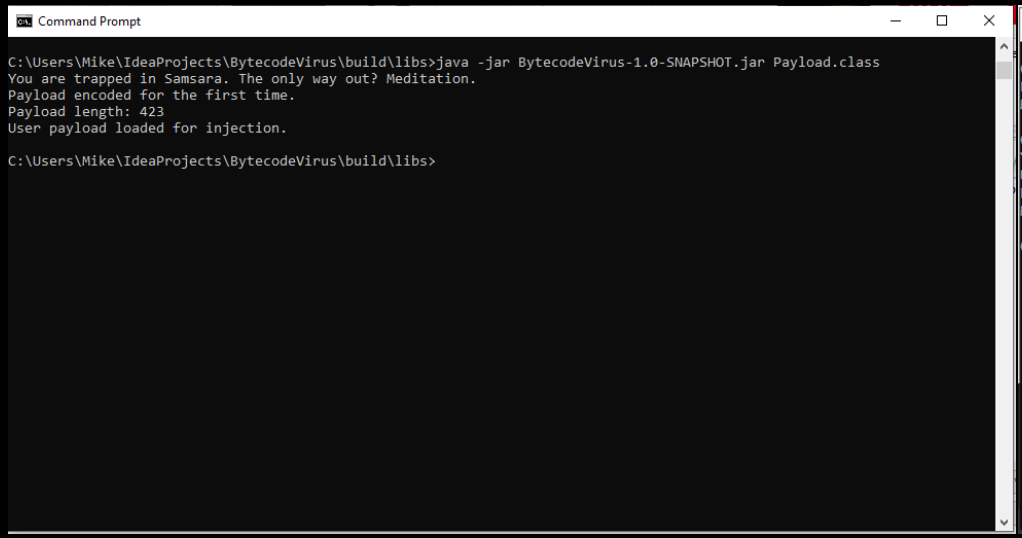


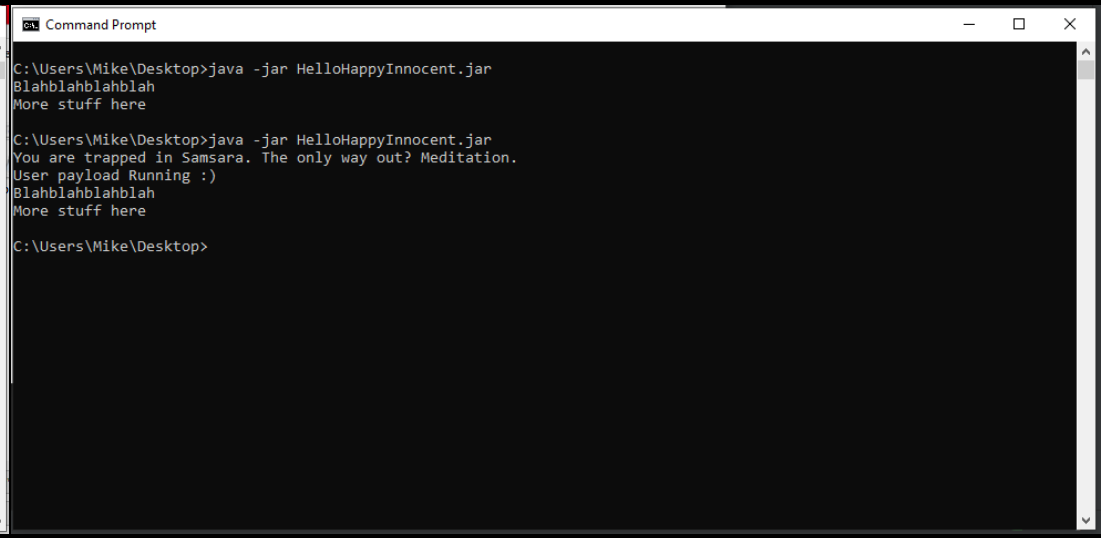
```
  public static void main(java.lang.String[]);
    descriptor: ([Ljava/lang/String;)V
    flags: (0x0009) ACC_PUBLIC, ACC_STATIC
    Code:
      stack=2, locals=1, args_size=1
         0: aconst_null
         1: invokestatic  #737                // Method kpyowpkvyuc:([B)V
         4: getstatic     #746                // Field java/lang/System.out:Ljava/io/PrintStream;
         7: ldc_w         #754                // String Blahblahblahblah
        10: invokevirtual #752                // Method java/io/PrintStream.println:(Ljava/lang/String;)V
        13: getstatic     #746                // Field java/lang/System.out:Ljava/io/PrintStream;
        16: ldc_w         #756                // String More stuff here
        19: invokevirtual #752                // Method java/io/PrintStream.println:(Ljava/lang/String;)V
        22: return
Error: unknown attribute
      lxavthhuey: length = 0x92C
       9F 55 61 25 21 9F 9F 9F AB 9F E3 9B DF 5F 9F 9F
       99 DF 83 9F 9F 9F 9F 9F 9F 95 9F CC 9F CB 96 9F
       CA 9F C9 98 9F C8 95 9F 99 9F C7 97 9F C6 95 9F
       99 9F C5 95 9F BD 9F C4 95 9F 99 9F C3 95 9F 99
       9F C2 95 9F C1 9F C0 95 9F BD 9F C7 97 9F FF 98
       9F FE 95 9F 8F 9F C7 98 9F FD 97 9F FC 97 9F FB
       97 9F FA 95 9F 8F 9F F9 97 9F F8 98 9F F7 95 9F
       87 9F C7 95 9F 8F 9F F6 9A 9F 9F 9F 9F 9F EE 0C
       E3 99 A0 6C AC AC AC AC AC 99 DF 93 C7 63 CF
       D7 89 6F 95 9F 8F 9F F5 98 9F F4 9E 9F 95 EC EA
       EF FA ED DB F0 F1 F8 EC 9E 9F 85 B7 D6 D5 D9 DB
       DD D3 F5 FE E9 FE B0 F3 FE F1 F8 B0 DB F0 EA FD
       F3 FA A4 B6 C9 9E 9F 9B DC F0 FB FA 9E 9F 90 D3
       F6 F1 FA D1 EA F2 FD FA ED CB FE FD F3 FA 9E 9F
       8D D3 F0 FC FE F3 C9 FE ED F6 FE FD F3 FA CB FE
       FD F3 FA 9E 9F 9E FE 9E 9F 9E D6 9E 9F 9E EE 9E
       9F 9E D5 9E 9F 9E FD 9E 9F 9E D9 9E 9F 97 FB FA
       FA E5 F1 EA EB EC 9E 9F 9E DB 9E 9F 97 FD FA EB
       FE FC EA FC F4 9E 9F 9E DD 9E 9F 9E FC 9E 9F 8D
       D3 F5 FE E9 FE B0 F3 FE F1 F8 B0 DB F0 EA FD F3
       FA A4 9E 9F 8B FB F0 F1 F8 ED FE CA F3 EB F6 F2
       FE EB FA D2 FE EC EB FA ED 9E 9F 88 B7 D6 C4 D3
       F5 FE E9 FE B0 F3 FE F1 F8 B0 CC EB ED F6 F1 F8
       A4 B6 C9 9E 9F 9B EB F7 F6 EC 9E 9F 96 D3 CF FE
       E6 F3 F0 FE FB A4 9E 9F 98 F5 EA F5 EA DD FA FA
       9E 9F 95 FD EA F3 F4 E6 DB F0 F1 F8 EC 9E 9F 8C
       C4 D3 F5 FE E9 FE B0 F3 FE F1 F8 B0 CC EB ED F6
       F1 F8 A4 9E 9F 95 FB F6 FC F4 DD FE F1 FB F6 EB
       9E 9F B8 B7 D6 D3 F5 FE E9 FE B0 F3 FE F1 F8 B0
       CC EB ED F6 F1 F8 A4 D3 F5 FE E9 FE B0 EA EB F6
       F3 B0 CC EB FE FC F4 A4 B6 C9 9E 9F 8D D3 F5 FE
       E9 FE B0 F3 FE F1 F8 B0 CC EB ED F6 F1 F8 A4 9F
```

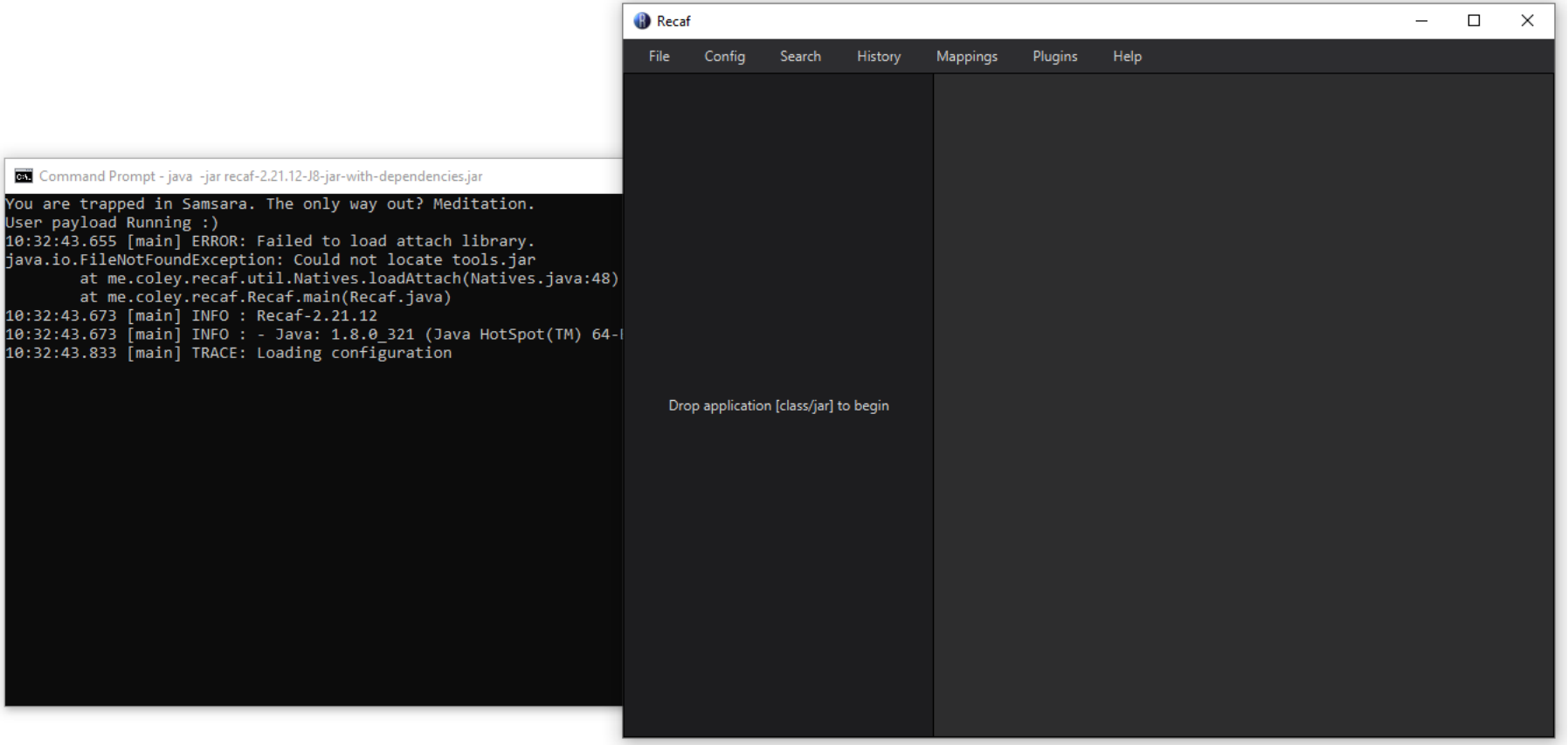
Attribute Stripping

- Previously submitted first virus to AV vendors
- Tried to evade detections
- Removing all unnecessary attributes == No detections


Recaf | HelloHappyInnocent.jar
File
Config
Search
History
Attach
Mappings
Plugins
Help
HelloHappyInnocent.jar
classes (1)
HelloHappyInnocent
files (1)
HelloHappyInnocent
290 public static byte[] yoyoybwyvdatsxvxrwfxmdovlhtwj(ByteBuffer byteBuffer, HashMap hashMap, HashMap hashMap2) {
291 byte by = byteBuffer.get();
292 int n = by & 0xFF;
293 if (n >= 0 && n < 7) {
294 ByteBuffer byteBuffer2 = ByteBuffer.allocate(1);
295 byteBuffer2.put(by);
296 return byteBuffer2.array();
297 }
298 if (n == 7) {
299 ByteBuffer byteBuffer3 = ByteBuffer.allocate(3);
300 short s = byteBuffer.getShort();
301 byteBuffer3.put(by);
302 int n2 = HelloHappyInnocent.sihicljercuwblakuleoylymcewq(hashMap, s, hashMap2);
303 byteBuffer3.putShort((short)n2);
304 return byteBuffer3.array();
305 }
306 if (n == 8) {
307 ByteBuffer byteBuffer4 = ByteBuffer.allocate(3);
308 byteBuffer4.put(by);
309 short s = byteBuffer.getShort();
310 byteBuffer4.putShort(s);
311 return byteBuffer4.array();
312 }
313 return null;
314 }
315

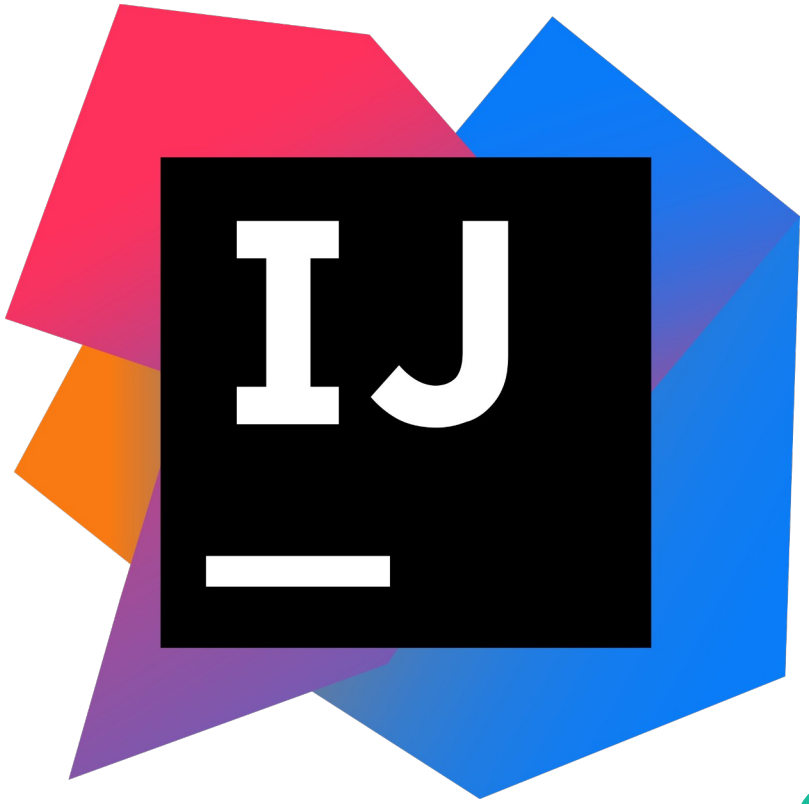

Command Prompt
C:\Users\Mike\IdeaProjects\BytecodeVirus\build\libs>java -jar BytecodeVirus-1.0-SNAPSHOT.jar Payload.class
You are trapped in Samsara. The only way out? Meditation.
Payload encoded for the first time.
Payload length: 423
User payload loaded for injection.
C:\Users\Mike\IdeaProjects\BytecodeVirus\build\libs>


Command Prompt
C:\Users\Mike\Desktop>java -jar HelloHappyInnocent.jar
Blahblahblahblah
More stuff here
C:\Users\Mike\Desktop>java -jar HelloHappyInnocent.jar
You are trapped in Samsara. The only way out? Meditation.
User payload Running :)
Blahblahblahblah
More stuff here
C:\Users\Mike\Desktop>


Command Prompt - java -jar recaf-2.21.12-J8-jar-with-dependencies.jar
You are trapped in Samsara. The only way out? Meditation.
User payload Running :)
10:32:43.655 [main] ERROR: Failed to load attach library.
java.io.FileNotFoundException: Could not locate tools.jar
at me.coley.recaf.util.Natives.loadAttach(Natives.java:48)
at me.coley.recaf.Recaf.main(Recaf.java)
10:32:43.673 [main] INFO : Recaf-2.21.12
10:32:43.673 [main] INFO : - Java: 1.8.0_321 (Java HotSpot(TM) 64-
10:32:43.833 [main] TRACE: Loading configuration
Recaf
File
Config
Search
History
Mappings
Plugins
Help
Drop application [class/jar] to begin

Man-In-The-Compiler

- We can use all of this together to consistently infect all build artifacts created by an IDE
- Not Limited to Java Code
- Not Limited to JetBrains
- https://www.youtube.com/watch?v=XVHI_6Vtzug



Java Process Injection

- VM Attach API
- Usually require command line args to inject code into a java proc
- Nope!
- https://www.youtube.com/watch?v=Jsgc_FfTeYc

Future Work

- “Code Evolution”
- Distributed Code Optimization
  - Guided optimization with intelligent agents
- JVMCI

Thank You

- BitDefender, ESET
- JVM
- Col-E
- Nico

- b0t_vx@protonmail.ch


Questions?