A Study of the Packer Problem and Its Solutions

Fanglu Guo Peter Ferrie Tzi-cker Chiueh

Presentation by Will Whiteside

What is a packer?

- Not a football player in Green Bay
- Packers are code compression/obfuscation engines that take arbitrary code as input and produce an excutable that is obfuscated, yet still produces the same results when run.

So, packers are a malware tool?

- No.
- UPX is an open source tool to compress executables that is used on many valid files.
- Portableapps.com hosts a large number of executables designed to run from a usb drive, most of these are compressed with UPX

What do we want to do then?

- We want to be able to scan the packed executable and determine if it is malicious.

How?

- Two ways
    - Unpack the executable then scan it using standard scanning tools (static)
    - Run the packed executable and interrupt it at some point to scan the image (dynamic)

Static Analysis

- Steps:
  - Recognize the packer – assign it to a packer family.
  - Identify a packer – determine version of packer.
  - Create an automated recognizer – create a program that performs the two above steps.
  - Create an unpacker – restore a packed binary to its unpacked form.

How long does this take

You get a new executable, run your recognizers against it to see if any of them recognize it. None of them do. Manually recognize and identify the packer, packer makers try to make this difficult. If the previous step is completed successfully, creating a recognizer is relatively simple. The packing procedure might well be irreversible, and even if it's not, producing a inverse function for it is likely NP complete.

So, that's like...

- 6 hours to 6 months per unpacker.
- There are 1200 packed samples in the Symantec backlog.
- That's between 7200 and 1,152,000 work hours by an experienced engineer, with an expected average on the order of 100,000 work hours
- Using a thoroughly reasonable average pay of $40/hr, that's $288,000 to $46,080,000 with an expected average of $4,000,000

I don't have that much money

- Neither do I.
- And creating a new packer version is not nearly as hard, and pays about as well as the other side.

So static analysis is really hard?

- You're a smart one.
- So let's try dynamic analysis.

So dynamic analysis?

- Let the program run, and at some point stop the execution and run the av program on the memory image.

That sounds easy

- That's why packer designers try to avoid letting you do it.
- You have to monitor the program somehow, and this monitoring can be detected.

Monitoring?

- Three ways:
  - Debugging (gdb or something similar)
  - Image dumping (dump the image regularly during execution)
  - Emulating (the whole environment)

Sounds good

- Packers try to check for these methods and defeat them.

So what do we need to do?

- Create an algorithm that can detect when we need to perform an AV scan, and on what.

Thus Justin

- Justin, the just-in-time AV Scanning algorithm.
- Haha
- Justin requires two assumptions:
  - The address space layout of the program embedded within a packed binary after it is unpacked is the same as that if the program is directly loaded into memory
  - the unpacker in a packed binary completely unpacks the embedded program before transferring control to it.

So, those assumptions

- Are reasonable, violating them requires much more work than not violating them, and possibly makes the packer unable to take a generic binary as input.
- Violating the second assumption is equivalent to polymorphic code that can evade detection without packing, so why pack it?

Ok, so how does it work

- Monitor the program through system calls, this method is much less detectable than other methods
- Scan the program image when:
  - A control transfer to a dynamically created/modified page occurs
  - The stack is similar to that when a program is just loaded into memory
  - The command-line input arguments are properly set up on the stack

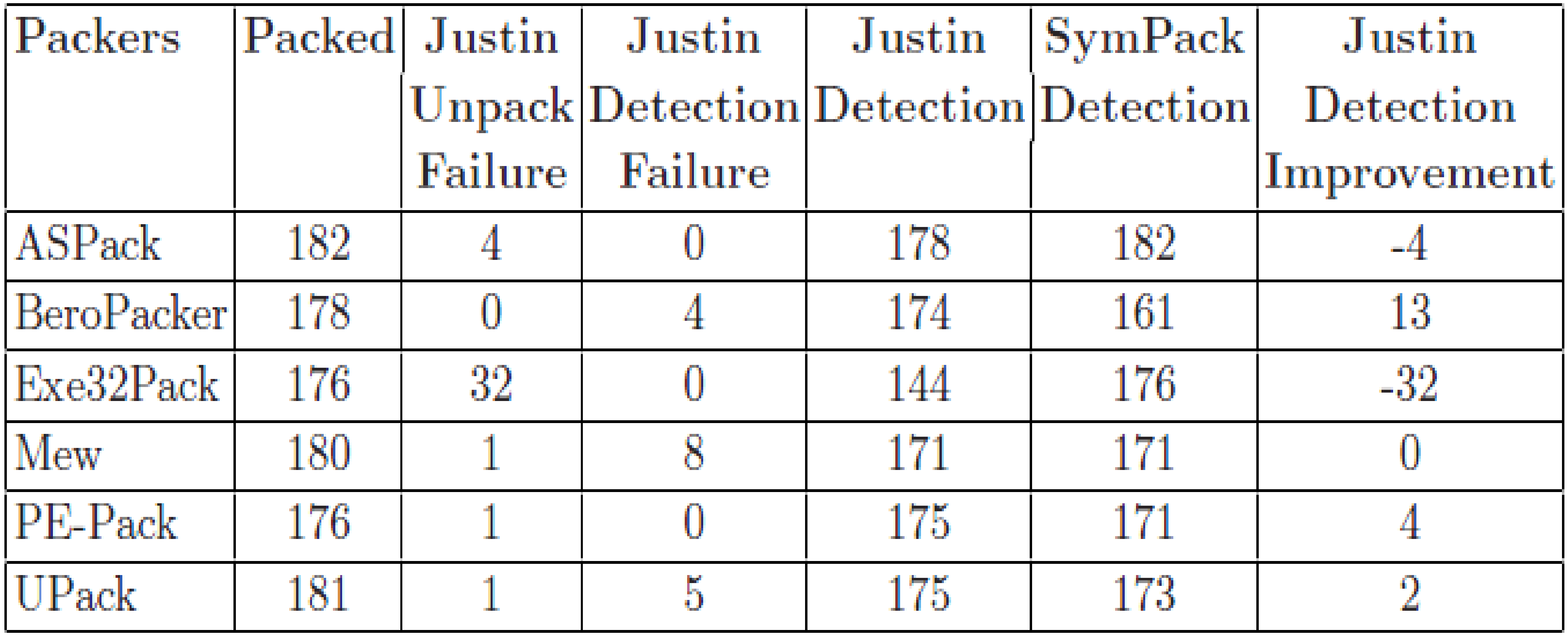
How?

- Monitor the writable and executable permissions of the memory pages
- Catch the exceptions here
- Dump the memory image and scan it for viruses

Implementation

- Justin is implemented for Windows only, but should extend to other architectures.
- Several potential holes are discussed, and they are plugged to my satisfaction

Results



Packers	Packed	Justin Unpack Failure	Justin Detection Failure	Justin Detection	SymPack Detection	Justin Detection Improvement
ASPack	182	4	0	178	182	-4
BeroPacker	178	0	4	174	161	13
Exe32Pack	176	32	0	144	176	-32
Mew	180	1	8	171	171	0
PE-Pack	176	1	0	175	171	4
UPack	181	1	5	175	173	2

That's pretty good

- But there's some failures there, why?
- Well, some of those failures are because the packed executable no longer runs.
  - Oh noes, the virus isn't detected because it can't be run, wait, it isn't running? I'm totally scared of it.
- Sometimes the packer doesn't do anything
  - Wait, your evil plan to obfuscate your program is to send it to me unmodified? Let's see how that works out for them, Cotton.

So most of those failures are harmless?

- In a word, yes.
- At least, according to the authors.

But that was a lot of work for not much results

- True, but that was on malware that we already have an unpacker for. Justin is designed to operate without an unpacker, and the true contribution is a scanner for packers that haven't been solved.

Ok, so what about those packers?

- Justin manages to unpack 12 of 13 packers, that's pretty good, but mostly relies on the packers not checking for Justin's API calls, an easy additional check.

So, Justin can be defeated

- Yes, and if Justin becomes an industry standard (or Justin like procedures are adopted) then these defeating measures will become an industry standard in the malware business.

So what else is wrong

- Justin might call for extra AV scans, making it take a long time to perform those scans.

That sounds bad

- Not really, the delay is minimal
- Their experiments show at worse, a 38% increase in delay, which on modern computers is probably less than a second on almost all programs
- Also, this only needs to be run once on each executable, and is usually faster than hard drive spin up times

Conclusions

- Justin provides a unpacking method that can unpack a large majority of current packers.
- Justin can be defeated, but provides a solid base for further research.