www.pwc.co.uk

SandGrox: Detecting and bypassing sandboxes

PwC Threat and Vulnerability Management

April 2017

Introduction

- Matt Wixey
- Threat and Vulnerability Management (TVM) – PwC
- Penetration tester and researcher
- Lead the TVM research function
- Previously worked in MPS SO leading R&D team

Sandboxes

- Lots of types
  - Virtualisation (VMs, isolated, & micro-virtualisation)
  - AV emulators
  - Hardware-based
  - Browser-based
  - etc

Sandboxes

- Detonate untrusted code in restricted environments
- May be used in conjunction with mail filters
- Or as part of host-based solutions
- And for malware analysis
- Bad news for pentesters ☹

Sandboxes

- Run code
- May place hooks on certain API calls
- Monitor actions the executable tries to perform
  - Registry read/writes
  - Creation of files
  - Remote connections
  - Suspicious strings
  - Evasion/detection activity

Sandboxes

- Changes/created files are usually not visible to other applications
- And usually not saved when the sandboxed application is terminated

The Problem

- Domains used for email and C2 get blacklisted ☹
- Malware is detected and blocked ☹☹
- Malware gets disassembled ☹☹☹
- AV vendors get signatures of our malware ☹☹☹☹

Solutions

- Sandboxes sometimes do a poor job at emulating genuine machines
  - CPU red pills
  - Network emulation
  - Fail to correctly emulate hardware
  - Less memory, smaller drives
  - Environmental artefacts
  - Plus, if we can actually get them to wait, we can still bypass them that way (delaying execution)

Arms Race - Example

- Malware introduces sleep calls to try and wait out the sandbox
- Sandbox forges system time or manipulates tick count
- Malware introduces acceleration detection mechanisms by taking time measurements before and after sleep calls
- Sandbox uses multipath exploration – execute both branches
- Malware uses differential reactions (server-side keys to decrypt)

Types of evasion

- Virtualisation-based (vendor-specific)
- Human interaction (e.g. mouse clicks)
- Environment (e.g. network)
- Configuration (e.g. file size, volume serial numbers)
- CPU red pills

Specifications

- Create a suite of checks to *detect* sandboxes
- In C++
- As lightweight as possible
- As vendor-independent as possible
- Checks should be weighted and return a total score
- Bypasses are a bonus

Limitations

- Hard to get hold of enterprise-grade sandboxes
  - *Hello, I'd like a trial of your product to try and hack it pls*
  - *Asking for a friend*
- Can't rely on big giveaways like "sboxie.dll" in memory, or "vbox" in registry ☹

Scope

- 3 months
- Windows only
- Suitable for compilation with MSVC++
- Outputs:
  - .cpp file
  - Compiled exe
  - Blog article
  - This presentation

Previous Research

- Pafish
- BlackHat ‘13: The Sandbox Roulette
- BlackHat ‘14: One Packer to Rule Them All
- BlackHat ‘16: AVLeak
- VMBuster
- InviZzzible
- EvilBunny (and lots of others!)
- …etc

EvilBunny

- Uses WMI to check for installed firewall and AV products
- Tries to detect known CPU red pills (process names)
- Also has a timing detection feature

Results

- 111 functions
  - *Sleep: 8*
  - *Memory: 5*
  - *Hardware: 14*
  - *Assembly: 33*
  - *Environment: 23*
  - *Filesystem: 12*
  - *Network: 10*
  - *Bypasses: 6*

SandGrox

- Neatly combines my love of bad puns with my love of obscure slang
  - And people said my English degree would be a waste of time
  - It was, though

SandGrox

- Combines 40 known techniques, publicly available in most cases, with 71 new checks (new AFAIK)
- Checks compiled into one executable, grouped into categories, and executed as a demo
- Results increment a variable with a weighting score if detected
- Weighted score returned

Deployment

- Single executable is for demo purposes only
- Intention is to identify the most reliable checks and incorporate them into our existing malware

Sleep

- Limited time & resources for dynamic analysis
- At some point, has to return execution to user/host
- 8 checks, all aimed at acceleration detection
- Using common API calls
- Not very successful (1 check was successful with 1 product)

Memory

- 5 checks, testing how well the sandbox handles big tasks
  - Hoping the sandbox says “malware pls” and falls over
  - Or at least fails to complete the task
- 2 checks both worked on 2 products
- 2 checks worked on 2 separate products (1 each)

Hardware

- 14 checks, looking at how well the sandbox emulates hardware devices and drivers
- 7 checks worked on 1-3 products

Assembly

- 33 checks looking at how well sandboxes cope with undocumented/privileged x86 instructions
- Historically this has been a big problem, particularly with AV emulators
- Plus instruction performance
- 2 checks worked on 3 products
- 1 check worked on 5 products

Environment

- 23 checks, mostly sanity checks
- But a lot of these will depend on how the environment is configured
- Ultimately we're looking for infections symptomatic of a non-corporate environment, so detections here don't necessarily mean we're in a sandbox
- 9 checks worked on 1-6 products

Filesystem

- 12 checks, testing how well the sandbox deals with file manipulation, plus some other sanity checks
- Again, may be configuration-dependent
- 8 checks worked for 1-6 products

Network

- 10 checks, looking at how sandboxes deal with network requests
- Sandboxes have 3 options:
  - Block network connectivity completely (easy to detect)
  - Emulate network connectivity (easy to detect)
  - Allow network connectivity, to softly troll us (can abuse this for other checks)
- SandGrox has checks to deal with all of these options ☺
- 6 checks worked on 1-6 products

Video demo 1

- SandGrox running on bare-metal machine

Video demo 2

- SandGrox running in a popular host-based sandbox

Results

- Tested on 6 products (AV sandboxes, host-based sandbox products, virtualised environments)
- Successfully detected all 6 products

	Bare metal	SB1	SB2	SB3	VM1	VM2	VM3
Score	3	21	28	27	44	47	37
Max	103	103	100	103	92	97	95
%	2.91%	20.39%	28%	26.21%	47.83%	48.45%	38.95%

Bypasses

- 6 functions
- Predominantly to do with delaying execution
- As well as bypasses for commercial sandbox products

Bypasses

- I’ll show you a (AFAIK) new method for delaying execution

Timelock Puzzles

Timelock Puzzles - Background

- Timed release crypto
- Solution is only discovered after a minimum period of time
- Goal is to “send information into the future”
- Really cool concept
- First proposed by Timothy May (of cypherpunks fame)

Timelock Puzzles - Applications

- Sealed bids
- Encrypted diaries
- Key escrow
- Time capsules

Timelock Puzzles - Example

Description of the LCS35 Time Capsule Crypto-Puzzle
by Ronald L. Rivest
April 4, 1999

As part of the celebration of the 35th birthday of MIT's Laboratory for Computer Science, LCS Director Michael Dertouzos will present an "LCS Time Capsule of Innovations" to architect Frank Gehry. The Time Capsule will reside in the new building, designed by Gehry, that will house LCS. The time capsule will be unsealed on the earlier of 70 years from the inception of the Laboratory (on or about 2033), or upon solution of a cryptographic puzzle, described herein. This puzzle is designed to take approximately 35 years to solve. It uses the ideas described in the paper

"Time-lock puzzles and timed-release Crypto"

by myself, Adi Shamir, and David Wagner. A copy of this paper can be found at http://theory.lcs.mit.edu/~rivest/RivestShamirWagner-timelock.ps.

The puzzle is designed to foil attempts of a solver to exploit parallel or distributed computing to speed up the computation. The computation required to solve the puzzle is "intrinsically sequential".

The problem is to compute 2^(2^t) (mod n) for specified values of t and n. Here n is the product of two large primes, and t is chosen to set the desired level of difficulty of the puzzle.

Timelock Puzzles - Background

- Could be used to evade AV / delay execution:
  - Encrypt payload
  - In decrypter stub, run timelock puzzle
  - After a certain period, the solution is found
  - Use solution as decryption key
  - Run decrypted bytes in memory, or write to new file and execute

Timelock Puzzles – Malware examples

- 1998 – IDEA.6155 virus – brute-forced decryption keys
- "Anti-Emulation Through Time-Lock Puzzles" (Ebringer, 2008)
  - PoC only, dual-architecture
- Rcrypt packer (Mohammed, 2014)
  - XXTEA (Corrected Block Tiny Encryption Algorithm)

Timelock Puzzles - Wixlock

- Using proof-of-work as a pseudo-timelock puzzle
- Specifically, a version of Bitcoin's mining PoW

Bitcoin ☹

- In a previous life, one of my areas of professional "expertise" was crypto-currency
- In January 2013, my boss asked me to write a short paper on it
- In July 2013 I gave him a 280-page report on Bitcoin's mechanics, roots in cypherpunk culture, and societal and economic implications
  - A+ for effort, F for following instructions

Bitcoin ☹

- To this day, not sure if anyone has actually read it
- Reward (punishment) was to spend the next 2 years:
  - Presenting on Bitcoin
  - Giving advice on Bitcoin
  - Writing tools to interact with Bitcoin
  - Sitting in meetings about Bitcoin
- On the plus side, I did make some money from Bitcoin ☺

Bitcoin ☹

- Current job does not involve Bitcoin
- Ironically I am now talking about Bitcoin of my own volition
- Why am I talking about it?
- Bitcoin mining is superficially a timelock puzzle (it has the same effect but isn't one, strictly speaking)

Mine, mine, mine, mine

- Bitcoin mining is based on a concept called proof-of-work
- Adam Back – HashCash (1997)
- Bruteforcing a partial hash collision

Mine, mine, mine, mine

- At set intervals, miners download a list of recent transactions (which will become the next block in the blockchain)
- They take different inputs, and perform a double SHA-256 on them
- Aiming to generate a hash such that a substring of the hash matches a substring of the target

Mine, mine, mine, mine

- e.g. does the resulting hash have an equal or greater number of zeroes at substr(0,5), compared to substr(0,5) of the target string?

Mine, mine, mine, mine

- Take hash of prev block (H1), timestamp (T), nonce (N), hash of Merkle Root of current block, (H2), and other data (D)

```
do
{
   R = DSHA-256(H1 + T + N + H2 + D);
   N++;
}
while (!(R.substr(x,y)) == (target.substr(x,y)))
```

Mine, mine, mine, mine

Target = 0003ea03e6651a4f1c2f03528cd031ec

Nonce = 1

Result = 6f1ed002ab5595859014ebf0951522d9 **REJECTED**

Nonce = 2

Result = fd35c67c22844b2c9859e55fb4c852fa **REJECTED**

Nonce = 3

Result = 0005d30e49aff3a7400f729a2c3bb410 **ACCEPTED**

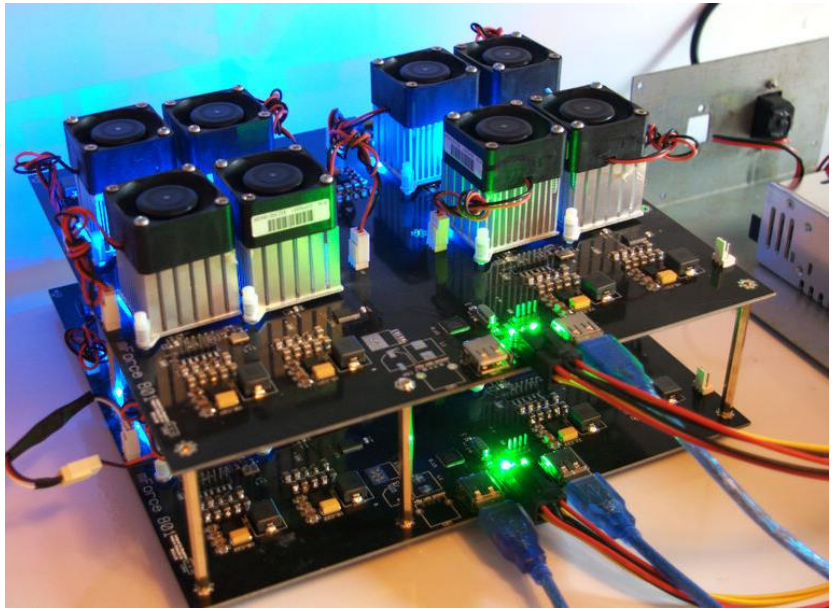
Mine, mine, mine, mine

- Nonce increments if attempt is unsuccessful
- Timestamps can also change, so nonces can be reused (ntwices?)
- Coinbase txn can also be changed, which changes H2
- Miners will perform this until a winning nonce is found, or until they have exhausted all possibilities

Mine, mine, mine, mine

- Target is a 256-bit number
- Substring is huge
- Computing it on a normal consumer CPU would take thousands of years, if not hundreds of thousands by now
- Hence mining pools, ASICs, etc

Mine, mine, mine, mine



Mine, mine, mine, mine

- Performance depends on hash rate
- Bitcoin protocol self-adjusts by aggregating time taken to hash a block
- Should take 10 minutes (2016 blocks every 2 weeks)
- If it's taken more or less than 10 minutes, the required difficulty changes accordingly

Mine, mine, mine, mine

- Not really a timelock puzzle
- Multiple nonces will give a correct solution
- So it’s more like a lottery

Mine, mine, mine, mine

- Every time you perform the hashing function, you have the same probability of winning
- Albeit that probability is incredibly small
- So don't logically have to start at 0 – could start anywhere
- Most winning nonces have been closer to 0, as by convention most miners have started there

PoW for sandboxes

- When the first (or second, or third, or *x*th) winning nonce is found:
- **use the resulting full hash as an AES key**
- **encrypt the malicious exe**
- **add encrypted bytes as a resource to stub**

PoW for sandboxes

- Perform same algorithm for the decryption stub
- As long as all variables and parameters are the same, the same hash will be generated
- Substring is known, but rest of the hash isn't
- Sandbox has no choice but to wait for it to finish as otherwise the bytes can't be decrypted

Coolness

- Sandbox can try nonces in sequence, randomised nonces, or start off at huge numbers – it makes no difference
- Because each nonce is equally likely to win
- And there are multiple nonces which will win

Coolness

- So the sandbox might find a nonce, or many nonces, which satisfy the substr check
- i.e. which have 7 zeroes at the start (for example)
- BUT
- There is only one nonce which both satisfies the substr check AND results in a hash which decrypts the payload

Customisation

- Change length of substring for longer wait times
- Or specify that the 2nd, 3rd, or *n*th hash should be used, rather than the first one
- Can also use random nonces rather than incrementing – same effect

Customisation

- Only one downside – you have to wait the same amount of time for the hash to be generated, in order to encrypt the payload
- Meh – I used to be an English Literature student
- And I worked in the police
  - I have extensive experience with long periods of inactivity

Execution

- Once the winning hash has been found, decrypt the payload to a byte array
- Inject bytes into a suspended process using the RunPE method

RunPE

- Starts new instance of process in suspended state
- Malware deallocates process memory with NtUnmapViewOfSection
- Malware allocates memory for new code with VirtualAlloc
- Uses WriteProcessMemory to write data to memory allocated
- Changes address of entry point and base address
- ResumeThread

DEMO

- msfvenom calc payload
- Encrypt (short substring, ~30 secs)
- Add encrypted bytes to decryption stub as a resource
- Scan original MSF payload to check
- Scan new payload
- Send stub to victim
- Execute stub

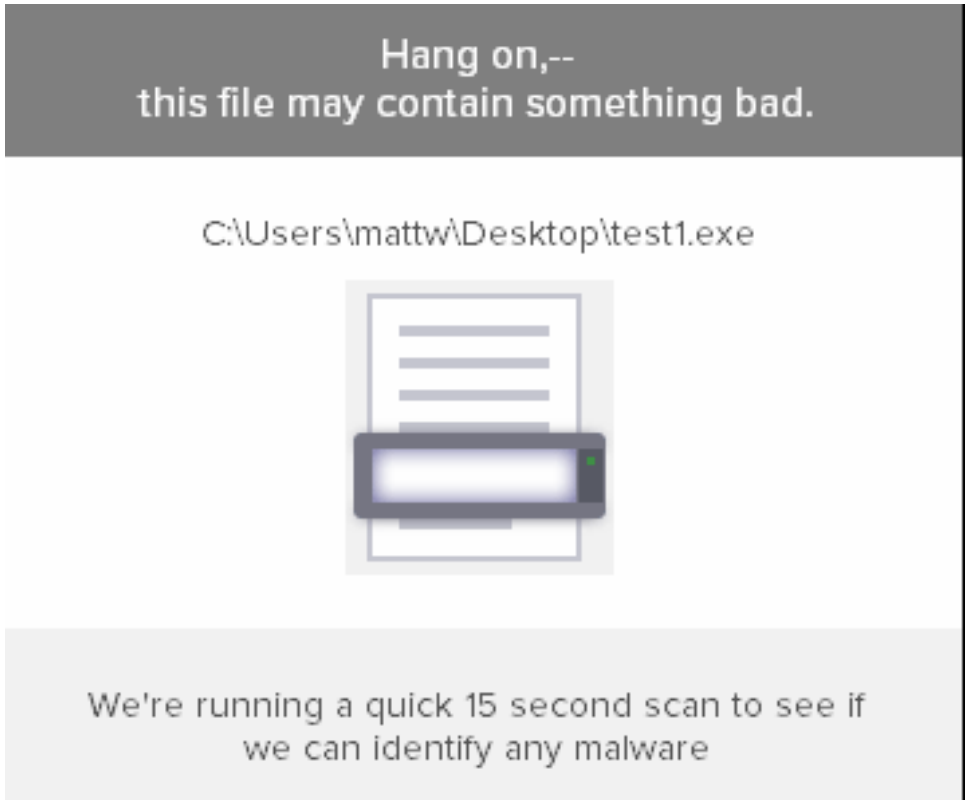
DEMO

Time delay success

- As expected, the original msfvenom payload is detected as malicious as soon as it's copied to the host

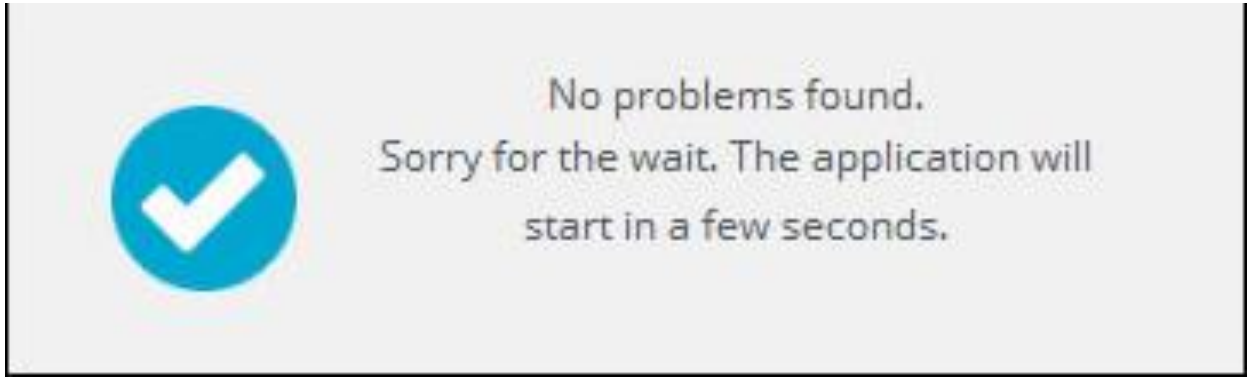
Time delay success

- Avast also suspects something is not right with the Wixlock exe, possibly recognising it as a decryption stub



Time delay success

- BUT
- Because of the time delay, we wait out the scan and everything is fine ☺



Buy an ASIC lol

Countermeasures

- Actually this wouldn’t work
- Commercial ASICs are hardwired to do a double SHA-256 hash of a string
- And are also designed to perform nonce incrementing
- ASICs are OTP, so can’t be repurposed

Countermeasures

- Could just use a different hashing algorithm anyway
- We don’t care which one is used
- As long as it still waits out the sandbox
- But it’s a fine balance between waiting too long (risk = user might turn off machine) and not waiting long enough

Further research

- Whitepaper
- Tests against more products
- Timelock
  - Combine acceleration checks e.g. if time to generate the key is significantly more or less than expected, corrupt the result so it won't decrypt
- Other timelock implementations e.g. factorisation

Further research

- Vendor-specific bypasses
- Other methods of evading acceleration detection
- Other options for malware upon sandbox detection
  - Providing false data
  - Attempting bypasses

Summary – sandbox detection

- Sandboxes still often provide a poor approximation of genuine machines
- Some are better than others
- I found a number of publicly available checks still worked on popular products
- Plus some new ones which also worked

Summary – sandbox detection

- Detection is relatively simple
- Especially if you just want your malware to terminate if it concludes it's in the matrix
- Bypasses are more difficult
- Execution delay is the most common, and there are a number of creative ways to do this

Get in touch!

Email: **matt.wixey@pwc.com**

Twitter: **@darkartlab**

Thank you!

Any questions?