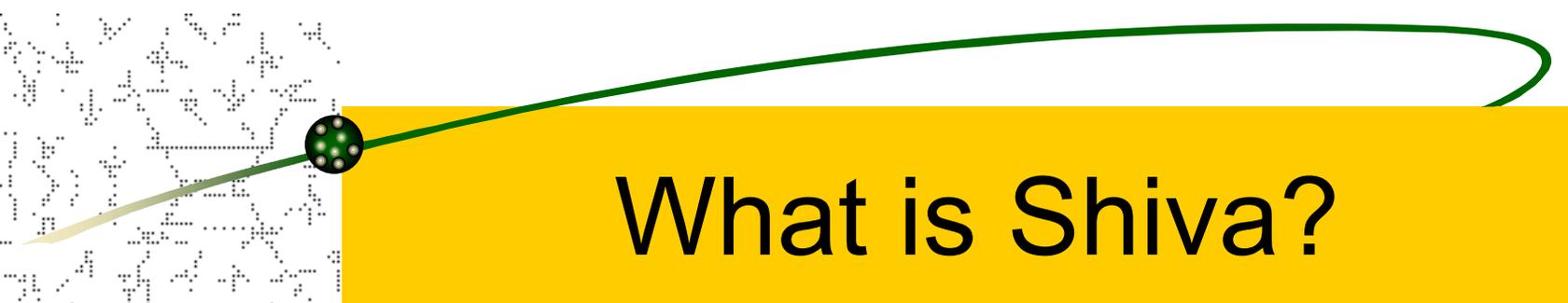




A Security Microcosm Attacking and Defending *Shiva*

Shiva written by Neel Mehta and Shaun
Clowes

Presented by Shaun Clowes
shaun@securereality.com.au



What is Shiva?

- ✦ Shiva is an executable encryptor
 - Encrypted executables run exactly as normal but are encrypted/obfuscated to make them much harder to reverse engineer or disassemble
- ✦ Resistant to analysis and modification
- ✦ Shiva works on Linux executables (in the ELF format)



ELF

- ✦ Executable and Linkable Format
- ✦ Used on virtually all modern Unix platforms
- ✦ Very descriptive and flexible format
 - Good for debuggers, compilers
 - As good for reverse engineers, executable patching and modification



The Field

- ✦ Executable encryption has been around for a long time
 - Since the late '80s
- ✦ Largely confined to the MS-DOS and Windows world
 - There are quite a number of *commercial* encryptors for windows



The Field

- ✦ Only recently been any work in the Unix field:
 - Burneye by Scut (2001)
 - ELFcrypt by JunkCode
 - UPX now runs on Linux



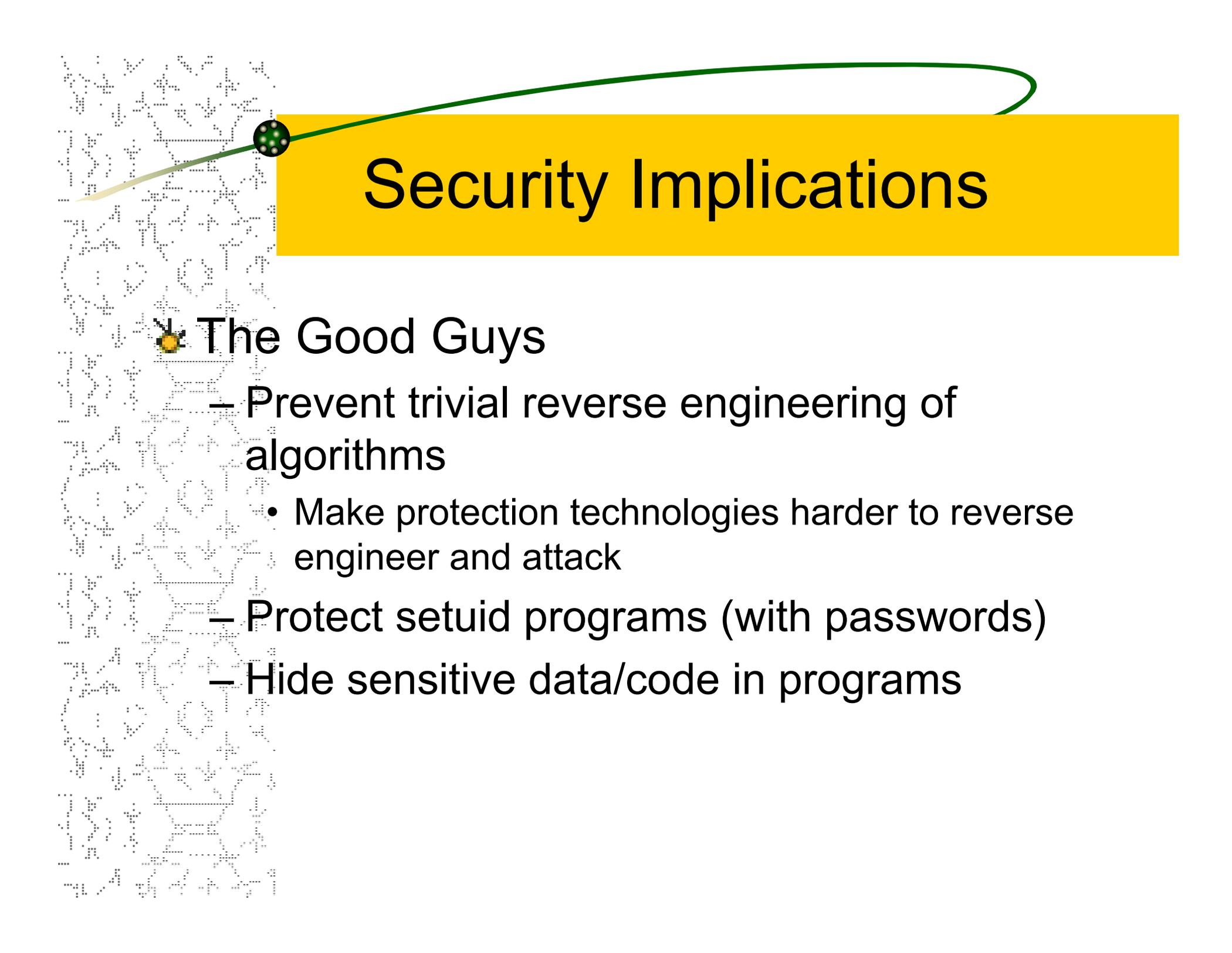
Our Goal With Shiva

- ✦ To provoke new research and development in, and wider understanding of:
 - Reverse Engineering
 - Binary manipulation



Advancements

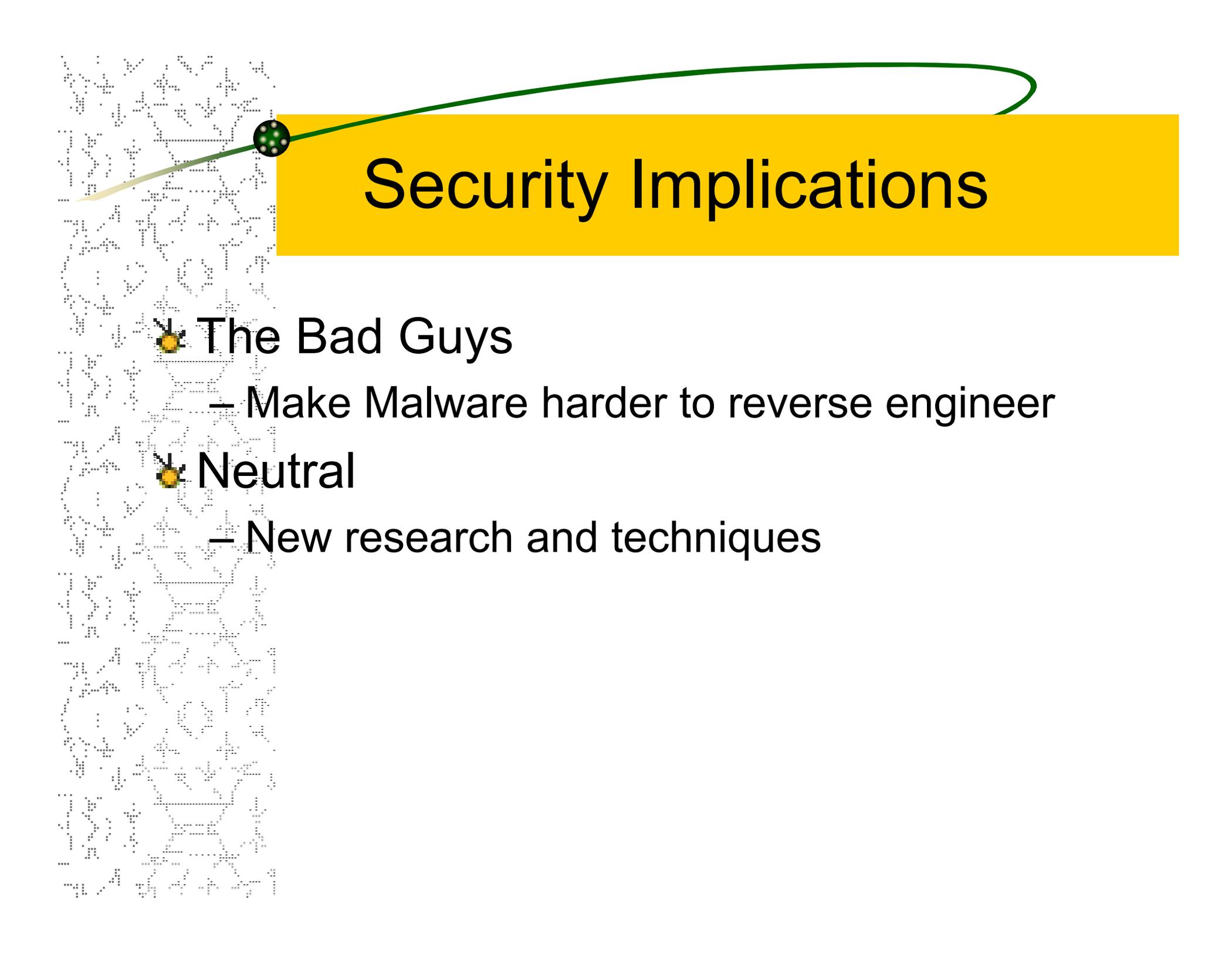
- ✦ Shiva brings many techniques from the Windows world to the Unix world
- ✦ Shiva also introduces some new techniques



Security Implications

✦ The Good Guys

- Prevent trivial reverse engineering of algorithms
 - Make protection technologies harder to reverse engineer and attack
- Protect setuid programs (with passwords)
- Hide sensitive data/code in programs



Security Implications

✦ The Bad Guys

- Make Malware harder to reverse engineer

✦ Neutral

- New research and techniques



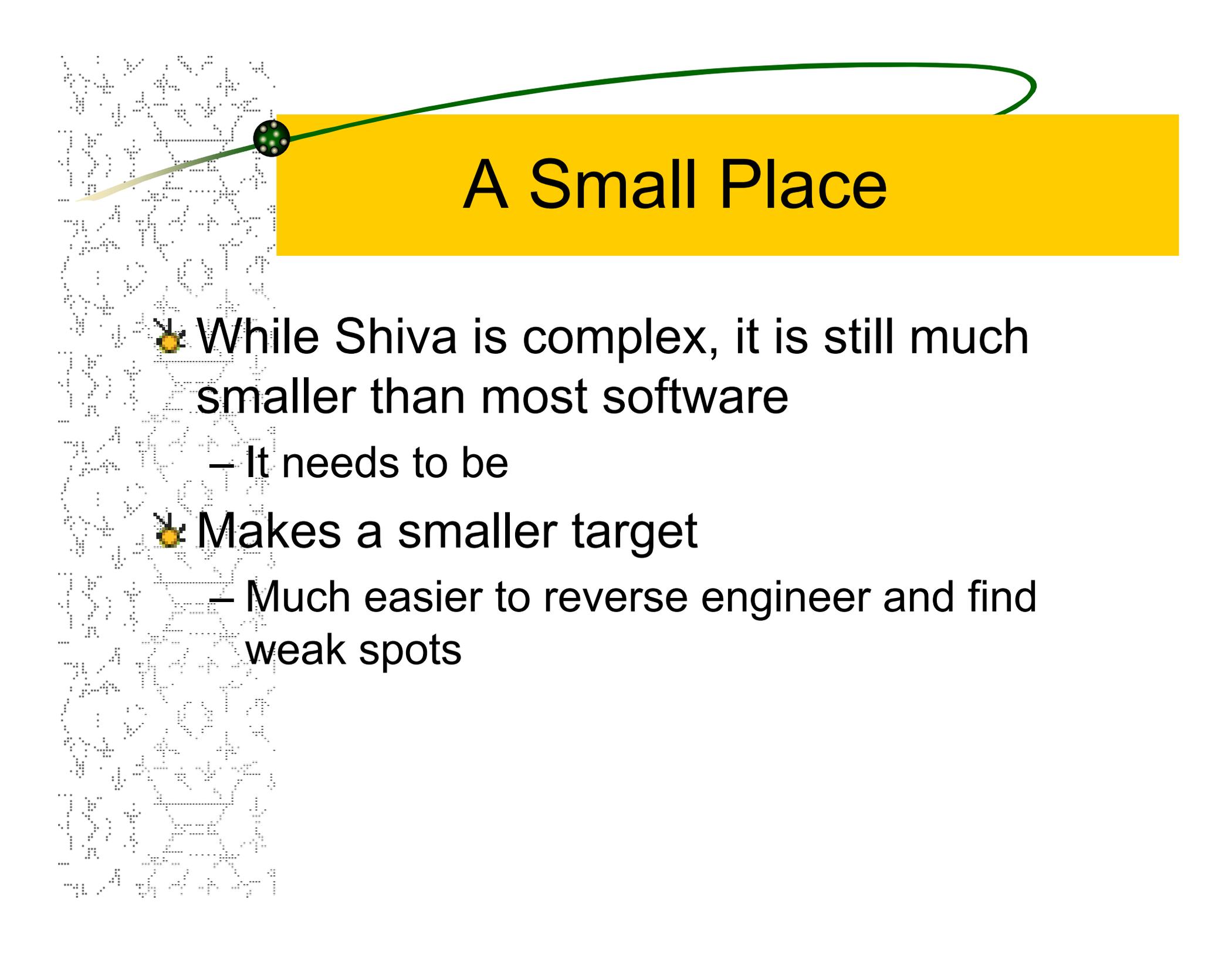
Shiva as a Microcosm

- ✦ Shiva is a protection technology
 - It protects a binary image from analysis or modification
 - Conceptually like any other protection technology, e.g a firewall, authentication scheme
- ✦ Attackers probe Shiva and it's output executables to find weaknesses



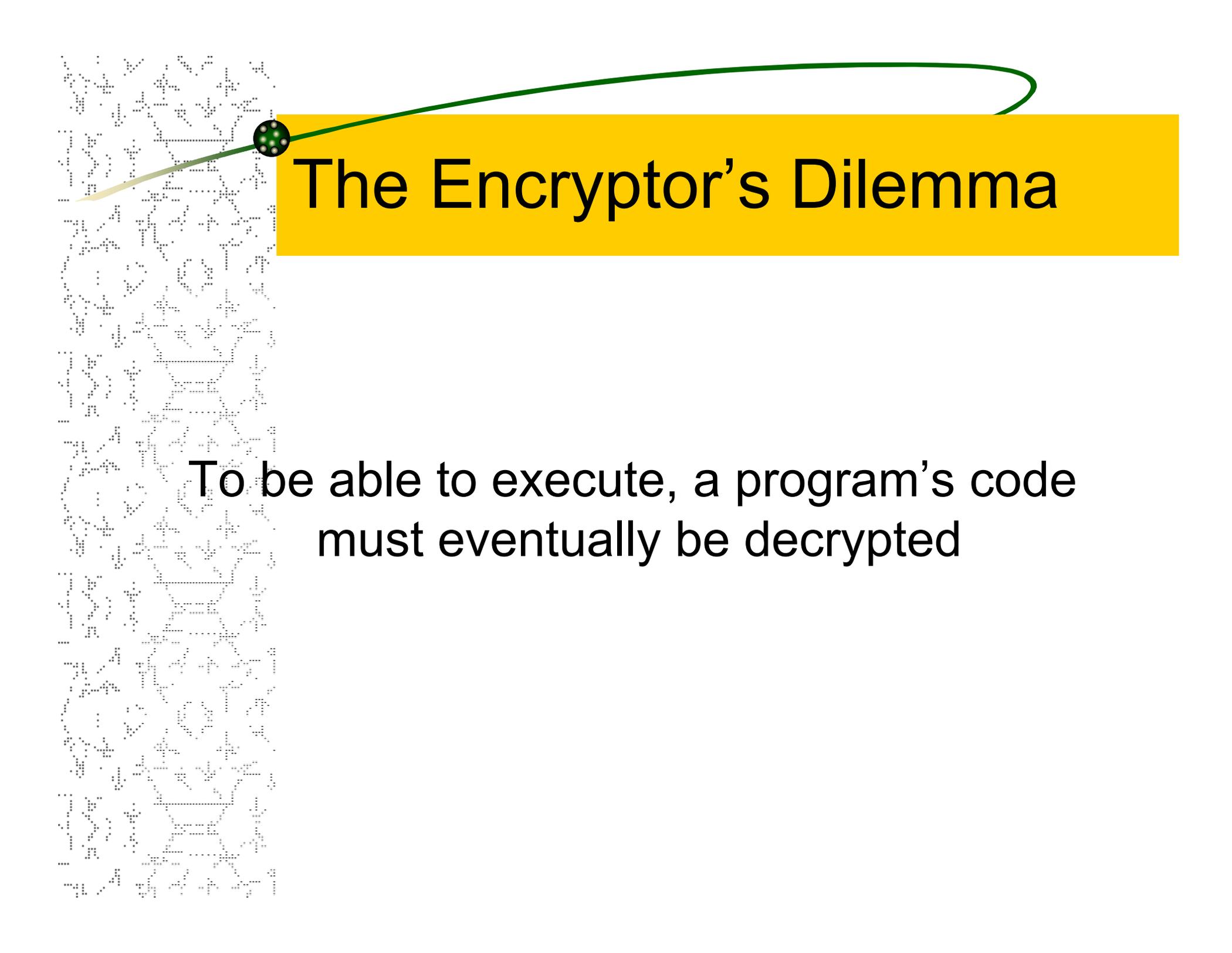
A Hard Place

- ✘ But Shiva is completely exposed:
 - Firewalls need to be probed blind
 - Shiva runs in an environment that can be **completely** controlled by an attacker
 - Right down to operating system behaviour
 - Even worse, we're telling everyone the details



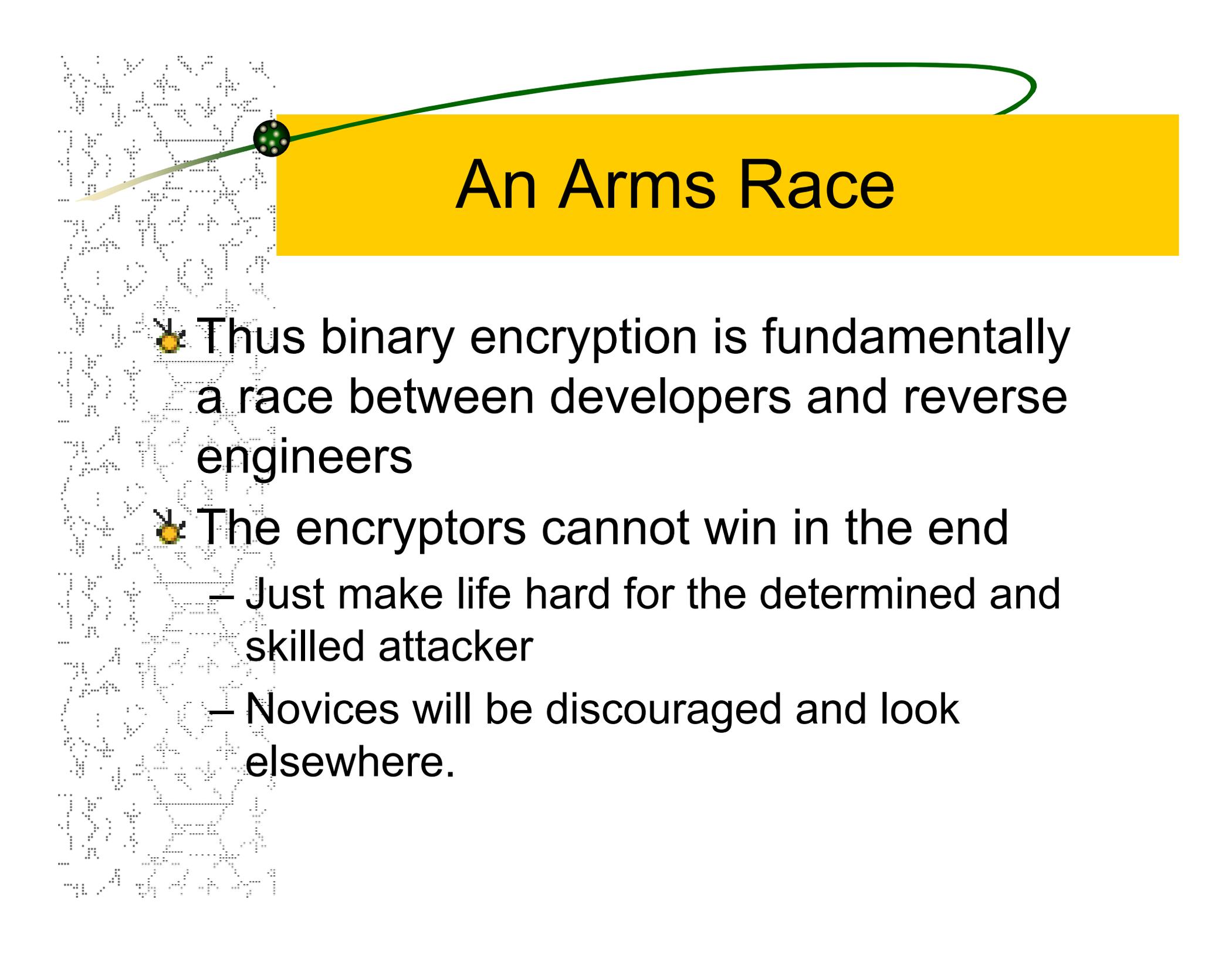
A Small Place

- ✦ While Shiva is complex, it is still much smaller than most software
 - It needs to be
- ✦ Makes a smaller target
 - Much easier to reverse engineer and find weak spots



The Encryptor's Dilemma

To be able to execute, a program's code must eventually be decrypted



An Arms Race

- ✘ Thus binary encryption is fundamentally a race between developers and reverse engineers
- ✘ The encryptors cannot win in the end
 - Just make life hard for the determined and skilled attacker
 - Novices will be discouraged and look elsewhere.



Encryption Keys

- ✦ If the encrypted executable has access to the encryption keys for the image:
 - By definition a solid attack must be able to retrieve those keys and decrypt the program
- ✦ To reiterate, binary encryption can only *slow* a determined attacker



Standard Attacks

- ✦ A good encryptor will try to deter standard attacks:
 - strace – System Call Tracing
 - ltrace – Library Call Tracing
 - fenris – Execution Path Tracing
 - gdb – Application Level Debugging
 - /proc – Memory Dumping
 - strings – Don't Ask



Deterring Standard Attacks

✦ strings

- Encrypting the binary image in any manner will scramble the strings



Deterring Standard Attacks

✦ ltrace, strace, fenris and gdb

- These tools are all based around the ptrace() debugging API
- Making that API ineffective against encrypted binaries is a big step towards making them difficult to attack



Deterring Standard Attacks



/proc memory dumping

- Based on the idea that the memory image of the running process must contain the unencrypted executable
- A logical fallacy



A Layered Approach

- ✘ Static analysis is significantly harder if the executable is encrypted on more than one level
- ✘ The layers act like an onion skin
- ✘ The attacker must strip each layer of the onion before beginning work on the next level



(Un) Predictable Behavior

- ✦ Efforts to make encryptor behavior differ from one executable to another are worthwhile
- ✦ The less generic the methodology, the harder it is to create a generic unwrapper



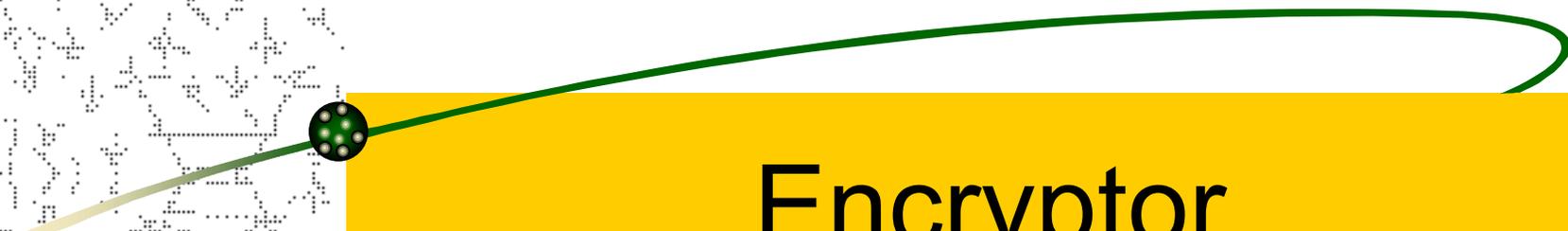
Shiva 0.97

- ✘ Currently encrypts dynamic or static Linux ELF executables
- ✘ Does not handle shared libraries (yet)
- ✘ Implements defences for all the attacks discussed so far



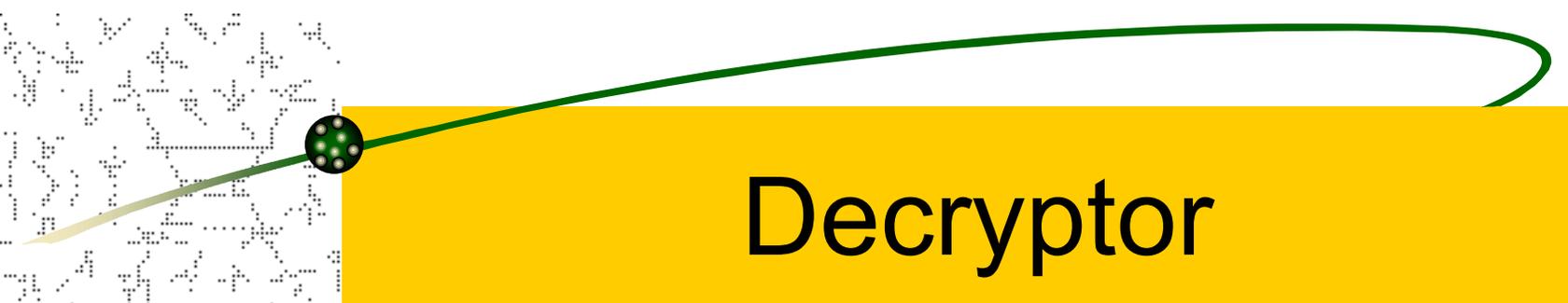
Encryptor / Decryptor

- ✦ Development of an ELF encryptor is really two separate programs
- ✦ Symmetrical operation



Encryptor

- ✦ Normal executable, which performs the encryption process, wrapping the target executable



Decryptor

- ✦ Statically-linked executable, which performs decryption and handles runtime processing
- ✦ Embedded within the encrypted executable
- ✦ Self contained
 - Cannot link with libc etc.



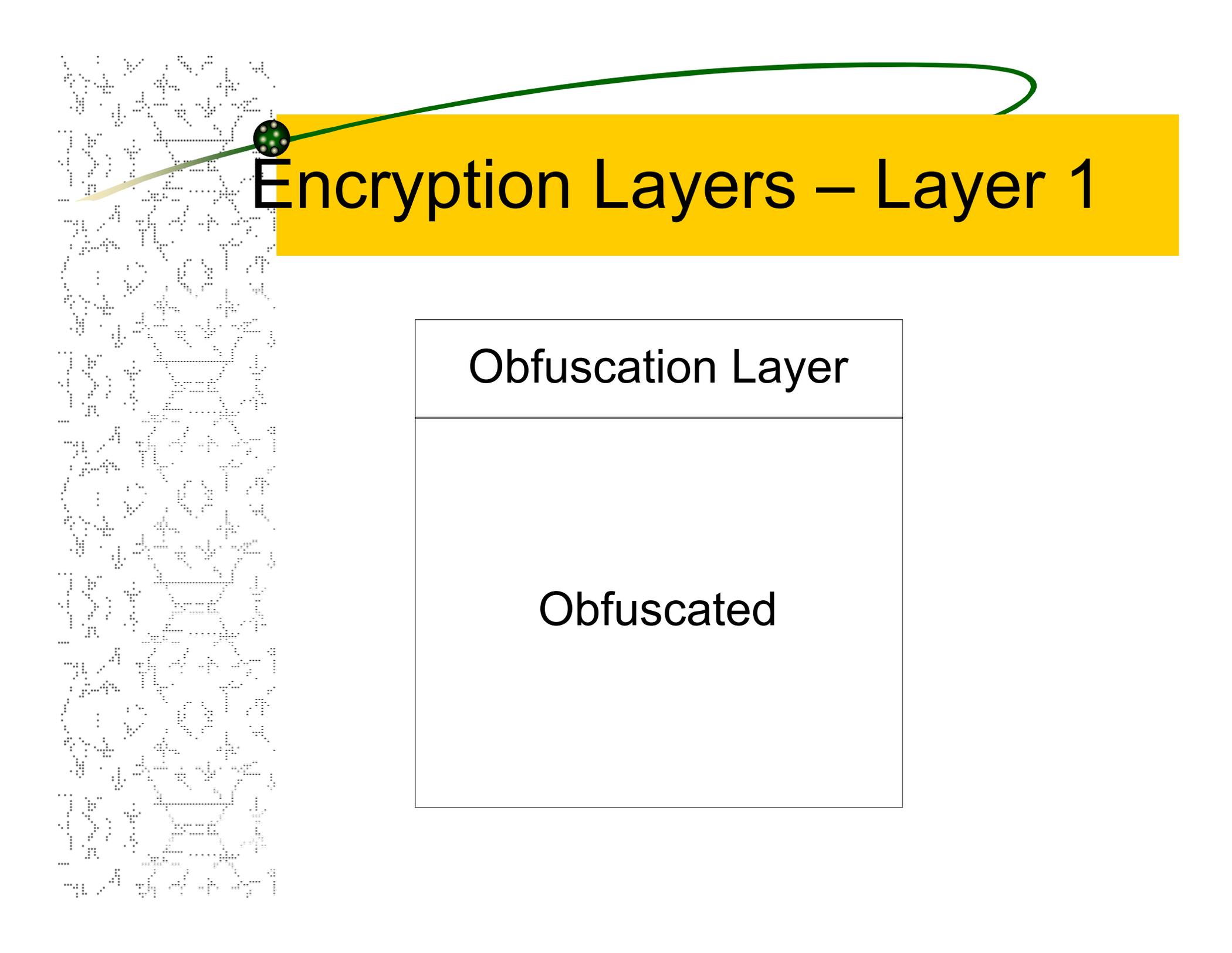
Dual-process Model (Evil Clone)

- ✦ Slave process (main executable thread) creates a controller process (the clone)
- ✦ Inter-pttrace (functional and anti-debug)



x86 Assembly Byte-Code Generation

- ✚ Allows for the generation of x86 assembly byte-code from within C (a basic assembler)
- ✚ Pseudo-random code generation, pseudo-random functionality



Encryption Layers – Layer 1

Obfuscation Layer
Obfuscated



Initial Obfuscation Layer

- ✦ Intended to be simple, to evade simple static analysis
- ✦ Somewhat random, generated completely by in-line ASM byte-code generation



Encryption Layers – Layer 2

Obfuscation Layer

Password Layer

AES Encrypted



Password Layer

- ✦ Optional
- ✦ Wrap entire executable with 128-bit AES encryption
- ✦ Key is SHA1 password hash, only as strong as the password



Encryption Layers – Layer 3

Obfuscation Layer

Password Layer

Crypt Block Layer

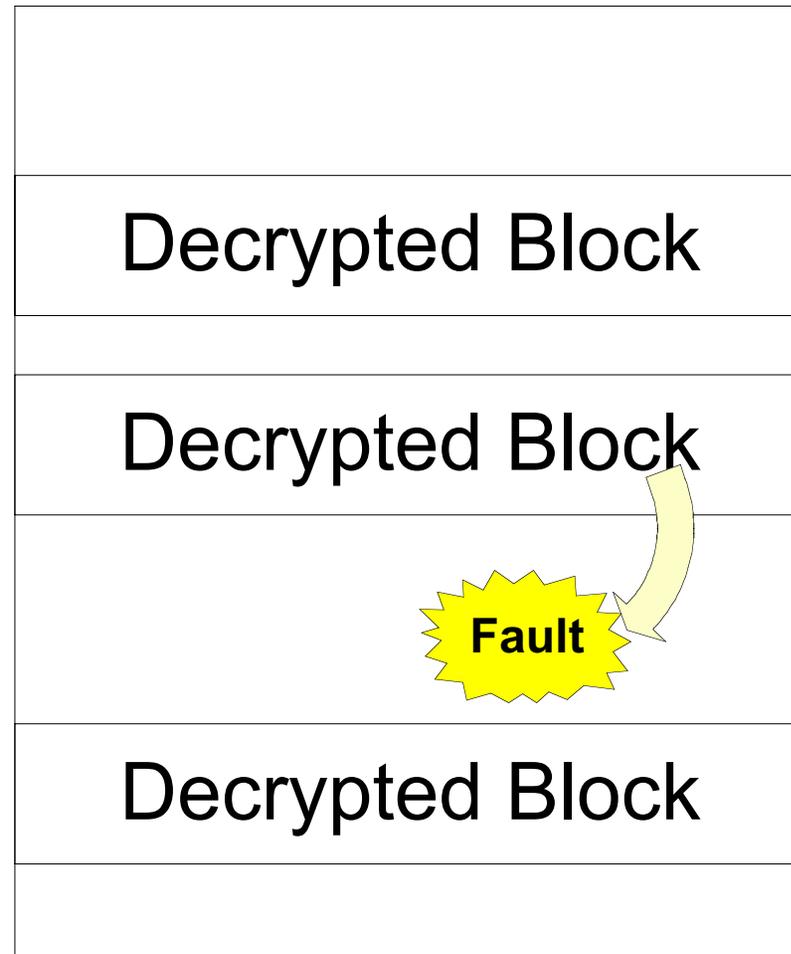
Crypt Blocks



Crypt Blocks

- ✦ Two important types – immediate map, map on-demand
- ✦ Controller process handles map on-demand blocks
- ✦ Random unmap
 - Only small portion of executable decrypted at any time
- ✦ Instruction length parsing – necessary to create map on-demand blocks

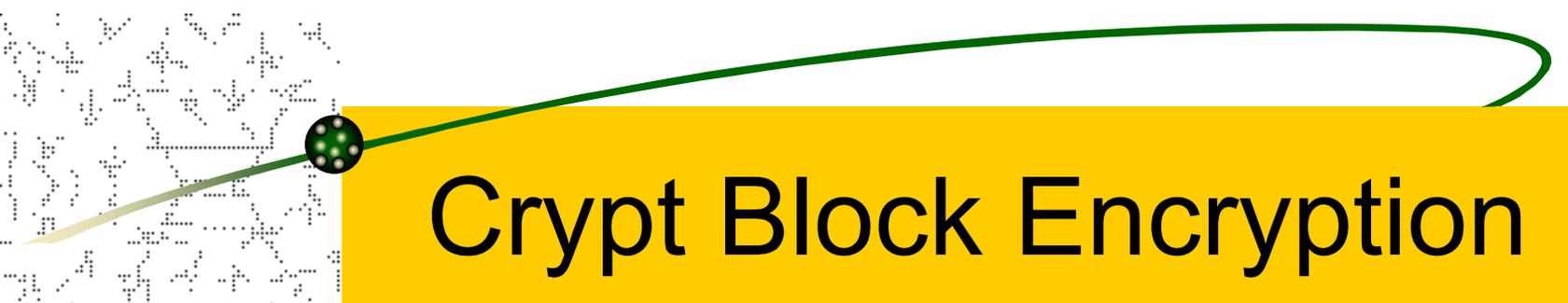
Crypt Block Mapping





Crypt Block Mapping

Decrypted Block
Cleared Block
Decrypted Block
Decrypted Block



Crypt Block Encryption

- ✚ Block content encrypted with strong algorithm
 - Guess
- ✚ Code to generate keys made pseudo-randomly on the fly (asm byte-code)
 - Keys are never stored in plain text
- ✚ Tries to bind itself to a specific location in memory (and other memory context)



Dynamically Linked ELF's

- ✦ Decryptor interacts with system's dynamic linker
- ✦ Decryptor must map dynamic linker itself, and then regain control after linker is done



Anti-debugging/disassembly

- ✦ Inherent anti-debugging provided by dual-ptrace – link verified
- ✦ Catch tracing:
 - Check eflags
 - Check /proc/self/stat



Anti-debugging/disassembly

- ✖ Timing and SIGTRAP
- ✖ Simple SIGTRAP catch
- ✖ JMP into instructions – common anti-disassembly trick



Problems Encountered, Solutions

- ✖ Clone, ptrace, and signals
- ✖ Fork processing
- ✖ Exec processing
- ✖ Life without libc
 - Simple implementations of malloc etc



Attacks Against Shiva

- ✦ We hoped Shiva would be defeated quickly
 - Turned out to be about three weeks before the first attack succeeded (A non public attack)
- ✦ We're now aware of three successful attacks against the previously released versions of Shiva



The First Attack

1. Allow the encrypted executable to execute but stop it after the first layer has executed (using ptrace)
2. Read the key routine locator block (at known location)
3. Execute the key routines **in process**
4. Use the keys to decrypt the blocks in memory



Exploited Weaknesses

- ✘ Reverse engineering showed that a lot of useful information was at fixed locations
- ✘ The first layer is weak
- ✘ The key routines are tightly coupled to the process image **but not** the control flow



The Second Attack

- ✖ Not sure of many of the details
- ✖ Involved a *complete* reverse engineering of the shiva loader
 - Including its libc



Shiva 0.96

- ✦ Released at BlackHat USA 2002
- ✦ Added code emulation functionality
- ✦ Requires significant code analysis.
 - Instruction by instruction processing
 - Function recognition, code flow analysis
 - Requires a fairly well designed and implemented framework



Instruction Emulation

- ✦ Easily accomplished via manipulating ptrace register structures
- ✦ Virtually every instruction can be emulated if its operation is understood



The Third Attack

- ✦ Executed by Chris Eagle
- ✦ Presented at BlackHat Federal 2003
- ✦ A novel hybrid static analysis approach
 - Emulating code execution via a plugin to IDA Pro
 - Can remove a lot of the tedious aspects of unwrapping protected code
 - ***Uber cool***



The Third Attack

1. Load ELF program data into a “virtual” environment
2. Emulate the execution of the first layer
3. Find the key headers and emulate them to retrieve the keys
4. Decrypt the blocks
5. Find the code emulation blocks and reapply them



Exploited Weaknesses

- ✘ Predictable locations
- ✘ The first layer is weak
- ✘ We certainly didn't predict emulators



Improving Shiva

- ✖ Remove some of the predictability
- ✖ Make it less of a sitting target
- ✖ Unwrappers resemble exploits
 - They're often fragile and dependent on hardcoded locations and values



Scrambling the Path

- ✦ For the encryptor to be able to randomize the loader it needs to store meta data
 - This is a weakness since a complete reverse of the encryptor would yield the meta data form
 - The meta data would help the attacker generate generic attacks on known invariant bits of the loader



Software as a Service

- ✚ This release of Shiva is now also a service
- ✚ Once a week a new version of Shiva is automatically uploaded to www.securereality.com.au/projects/shiva
- ✚ The loader is automatically post processed to make it less predictable



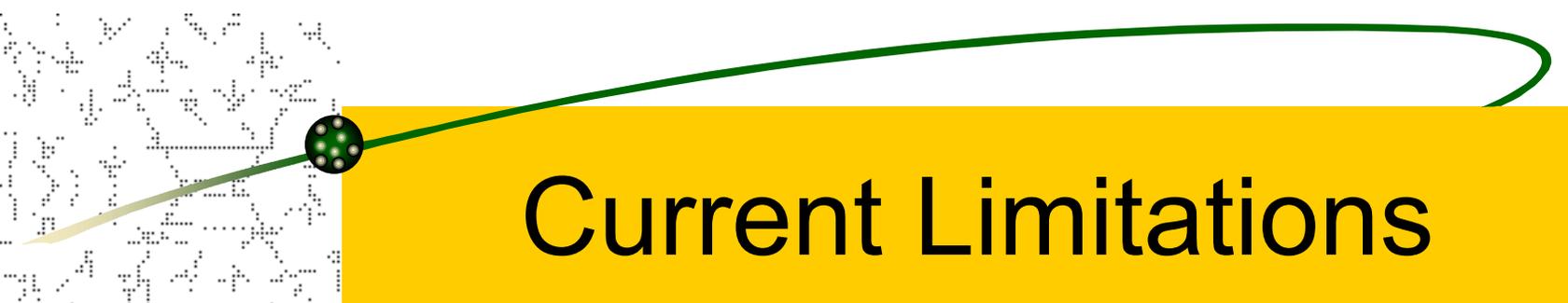
Morphing Code

- ✦ The current randomization engine is very simplistic, though it does remove predictable addresses entirely
 - Working on a full code flow analysis version
- ✦ The encryptor does perform some simple modifications of the loader too



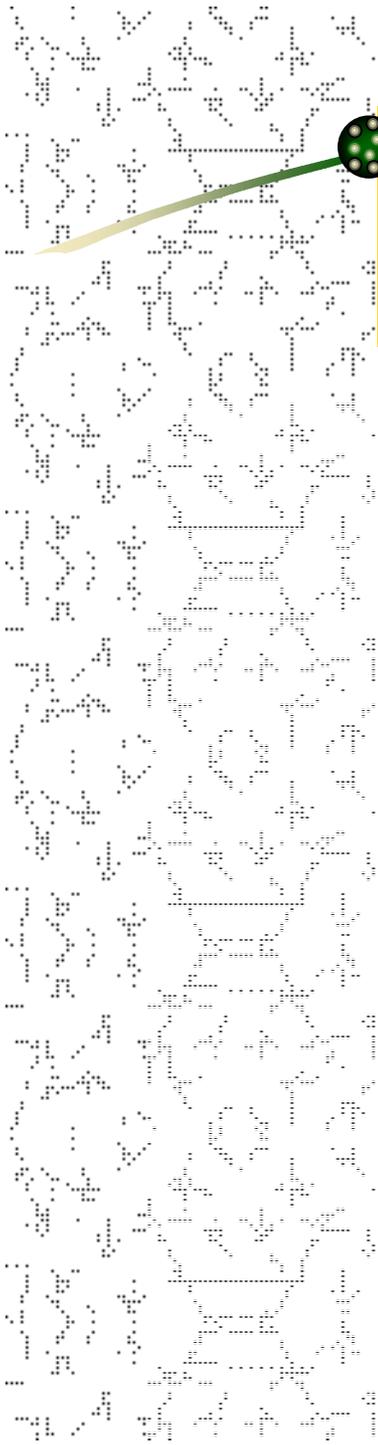
Development Pain

- ✘ Standard development approaches are anathema to an encryptor
 - Since they allow the reverse engineer to spot design patterns
- ✘ Makes developing Shiva painful
 - Trying to code in an undesigned fashion



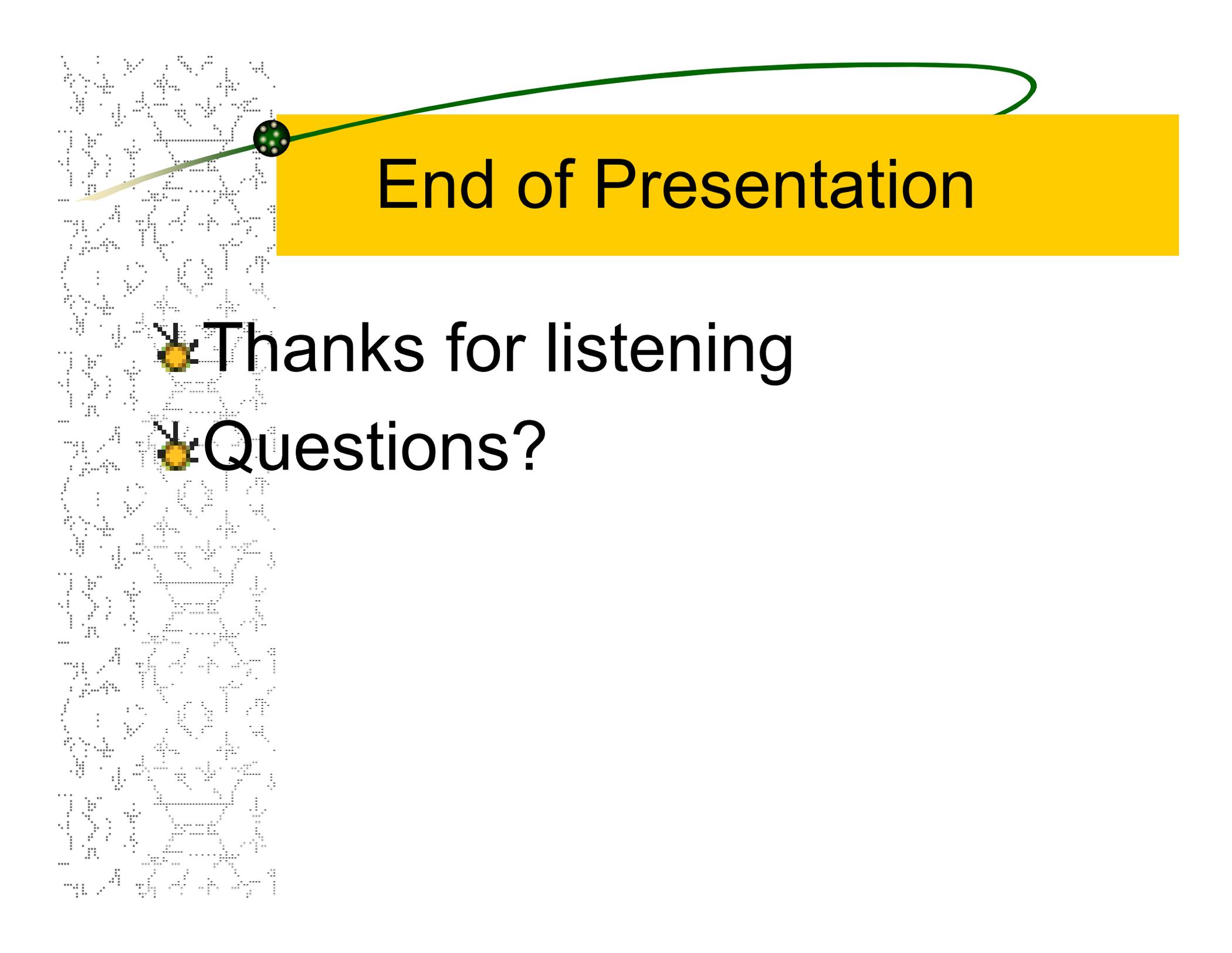
Current Limitations

- ✘ Can't handle `vfork()`, threads
- ✘ Can't encrypt static executables that call `fork()`
- ✘ On Linux, `exec()` fails if the calling process tries to exec a setuid program
- ✘ Section Headers
- ✘ Concentrating on deterring attackers 😊



Shiva in Action

Demo



End of Presentation

✿ Thanks for listening

✿ Questions?