# Context-Keyed Payload Encoding: Fighting the Next Generation of IDS


Dimitrios A. Glynos
dimitris *at* census-labs.com

Census, Inc.


Athens IT Security Conference (AthCon 2010)

# Overview

- Introduction
- Shellcode Detection Techniques
- Context-Keyed Payload Encoding
- Implementation
- Demonstration
- Best Practices
- Conclusion

# Introduction

# The Basics

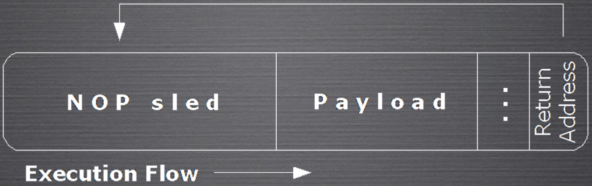
- What is shellcode ?
  - Memory corruption bugs sometimes allow an attacker to execute her own instructions on the CPU of a vulnerable host.
  - These instructions usually provide the attacker with a command interpreter (e.g. a UNIX shell) and that's why they're called *shellcode*.
- What is an Intrusion Detection System (IDS) ?
  - A system that detects malicious activities by examining a host's operating environment (HIDS) and/or network traffic (NIDS).
- This presentation focuses on:
  - Shellcode detection techniques for NIDS.
  - NIDS evasion techniques for stealthy shellcode.

# 5 Reasons for Tracking Shellcode on The Wire

- CVE-2007-1365 *OpenBSD IPv6 mbufs remote kernel buffer overflow*
- CVE-2007-2586 *Cisco IOS FTP Vulnerability*
- CVE-2009-0065 *Linux SCTP FWD Chunk Memory Corruption*
- CVE-2009-0950 *Apple iTunes ITMS Overflow*
- CVE-2010-0239 *Windows ICMPv6 Router Advertisement Vulnerability*

# Shellcode Detection Techniques

# Shellcode Environment




- Return Address
  - Points to an area close to the shellcode.
  - Overwrites a saved EIP or function pointer.
- NOP sled
  - Dummy instructions!
  - They guide the instruction pointer towards the actual shellcode when its address is not known in advance.
- Payload
  - Contains the shellcode instructions.

# THE 3 SCHOOLS OF MALWARE DETECTION

- Signature Matching
  - Detect known shellcode bytes [Snort]
  - Detect known NOP bytes (Snort thinks 25 ’C’s are a ’inc %ebx’ NOP sled)
  - Detect known return address ranges (Buttercup)
  - Cannot detect 0-day exploits.
- Anomaly Detection
  - Perform statistical analysis on traffic (Snort SPADE)
  - If incoming packets deviate from “normal” traffic/protocol, warn the user.
  - Requires training.
- Static / Dynamic Analysis
  - Inspect packets for code with certain characteristics (see [Polychronakis06]).
  - Takes time...

# Polymorphism and Metamorphism

- Polymorphic Encoding
  - Encrypt payload with random key.
  - Payload instructions will be decrypted and executed at runtime.
- Metamorphic Encoding
  - Reimplement a set of operations with equivalent instructions.
  - Build tools to generate the equivalent code automatically.

# The 3 Schools Revisited

- Signature Matching
  - Polymorphism allows the payload to evade detection.
  - Metamorphism allows the polymorphic decoder stub to evade detection.
  - See "Shikata Ga Nai" encoder of Metasploit.
- Anomaly Detection
  - Metamorphic encoders can produce instruction bytes that have similar statistical properties with the canonical traffic...
  - See "Alpha2" encoder of Metasploit.
- Static / Dynamic Analysis
  - Static Analysis fails to determine if a packet contains junk or a polymorphic payload.
  - Dynamic Analysis can spot the malicious payload, once it has emulated correctly the polymorphic code!

# EMULATION TROUBLES

- NIDSs guard the perimeter.
- NIDSs with emulation support, emulate incoming packets “blindly”.
- Emulation happens within a fake/minimal environment.
- What if the shellcode depends on a piece of information from the environment of the vulnerable host?
  - It will fail to execute on the NIDS.
  - But it may execute correctly on the vulnerable host.
  - Hmm, IDS evasion!

# Context-Keyed Payload Encoding

# The Main Idea

- Encrypt the payload with your favorite algorithm.
- At execution time, get the decryption key from the environment (context) of the vulnerable host!

# Memory-based Keying

- Use the bytes found at a specific memory location as the encryption key.
- |)ruid has implemented this for Metasploit.
  - To find memory addresses with static values, the tool `smem-map` is used.
  - See [ToorCon9] for more details.
- jDuck has written something similar, checking if a particular bit is set at a certain memory location.
- Can we guess this value for a remote host?
  - Think about distributions that use binary packages.
- PIE binaries and ASLR can be an issue here.

# CPU-based Keying

- The **cpuid** x86 instruction returns processor information.
  - Processor info is broken down into multiple *vectors*.
  - The number of available vectors depends on the processor model.
- R. R. Branco and Itzik use "Vendor ID" as a shellcode encryption key (see [Troopers09]).
- We will extend this to include all vectors containing Basic Processor Information.
  - XOR-ing all vector data gives us a richer 32-bit key.
- Can we guess this value for a remote host?
  - Think about standard server models and Qemu guests...

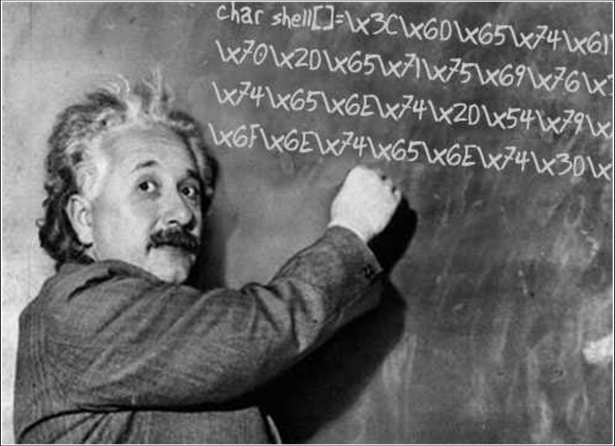
# Temporal Data-based Keying

- Build the decryption key from something that is going to be there for a certain amount of time.
- The Hydra shellcode engine (see [Hydra09]) uses some high-order bits from the time(2) system call.
- time(2) returns a 32-bit integer (secs since *epoch*).
- We'll use the 16 most significant bits, providing an execution window of 18 hours.
- Can we guess the time on a remote server? :-)
- Is it so difficult for this system call to be emulated?
- This encoder may slightly buffle reversers studying your code at a later time. But brute forcing 16bits is hardly a challenge...

# Filesystem-based Keying (new!)

- Usually NIDSs don't have access to the filesystems of the servers they are protecting.
- We can make a context key from filesystem metadata (see stat(2)).
- Good candidates: the st_size and st_mtime members of struct stat.
  - st_size is guessable if the target hosts a known software package in binary form.
  - st_mtime is guessable in Debian (records timestamp of last update by package maintainer).
  - Let's XOR these to create a context key!
- What happens if we stat(2) a file that we later rename/delete?
  - Is this temporal data? :-)

# Implementation



# Design Decisions

- Make CKPE a PenTester's Commodity.
  - Build on the Metasploit Framework!
- No Key Generator classes are available...
  - Each CKPE method becomes a separate encoder.
- Context Keys are generated by aux. applications.
  - Fed to CKP encoders via command line arguments.
- Actual payload encoder: Shikata Ga Nai, 32bit key.
- Execution in wrong context: Undefined behaviour :-)

*Usage example*

```
$ cd metasploit/trunk
$ ./tools/stat-key /bin/ps
0xbebaf012
$ ./msfpayload linux/x86/exec CMD=/bin/sh R > /tmp/raw_payload
$ ./msfencode -e x86/context_stat -t elf -i /tmp/raw_payload -o /tmp/encoded_payload \
STAT_KEY=0xbebaf012 STAT_FILE=/bin/ps
$ /tmp/encoded_payload
sh-3.2$
```

# Inside a Context-Keyed Payload Encoder

- A CKP encoder performs the following actions:
  1. Gets the context key from the user.
  2. Generates a context-key generator stub.
  3. Passes the key to a (polymorphic) encoder and generates the encoded payload.
  4. Returns the combination of stub and encoded payload to the user.
- To mix different stubs & encoders we need a standard way of passing the key to the encoder.
  - We use the eax register for this.
  - Compatible with existing Metasploit encoders.
  - Does not mess with stack / heap layout.

# The cpuid KeyGen Code

15 instructions, 32 bytes, 0x00 / 0x0a / 0xff clean!

| | |
|---|---|
| xorl %esi, %esi | zero out key register esi |
| xorl %edi, %edi | zero out loop iterator i (edi) |
| cpuid_loop: movl %edi, %eax | make i the $1^{st}$ cpuid parameter |
| xorl %ecx, %ecx | $2^{nd}$ cpuid parameter: always null |
| cpuid | |
| xorl %eax, %esi | XOR cpuid eax output with key |
| cmpl %esi, %eax | In $1^{st}$ iteration, esi = eax (dodgy!) |
| jne not_first_time | |
| leal 0x1(%eax, 1), %edi | $1^{st}$ iteration: i = last vector idx + 1 (anticipating bottom-of-loop decrement) |
| not_first_time: xorl %ebx, %esi | XOR the remaining registers with key |
| xorl %ecx, %esi | |
| xorl %edx, %esi | |
| subl $1, %edi | bottom-of-loop decrement: i = i - 1 |
| jne cpuid_loop | |
| movl %esi, %eax | place key in eax |

# THE TIME(2) KEYGEN CODE

*4 instructions, 10 bytes, 0x00 / 0x0a / 0xff clean!*

| | |
|---|---|
| xorl %ebx, %ebx | provide NULL argument to time(2) |
| leal 0xd(%ebx, 1), %eax | setup syscall number for time(2) |
| int $0x80 | execute the syscall |
| xor %ax, %ax | zero out the 16 least significant bits of the result |

# THE STAT(2) KEYGEN CODE

*13 instructions, 32 bytes + filename, 0x00 / 0x0a / 0xff clean!*

| | |
|---|---|
| fldz | fnstenv-style getPC |
| fnstenv -0xc(%esp) | |
| popl %ebx | |
| jmp over | jump over *filename* |
| *filename* | the filename |
| over: add $8, %ebx | get *filename* address in ebx |
| leal *filelen*(%ebx, 1), %edx | edx points after *filename* |
| xorl %eax, %eax | |
| mov %al, (%edx) | NUL-terminate *filename* |
| leal -0x58(%esp, 1), %ecx | make ecx point to new struct stat |
| mov $0xc3, %al | |
| int $0x80 | execute syscall stat(2) |
| movl 0x2c(%ecx), %eax | retrieve st_size member |
| xorl 0x48(%ecx), %eax | XOR st_size with st_mtime member |

# Demonstration

# Best Practices

# TIPS ON USING CKPE

- Use CKPE to hide your payload from automated malware detection tools (not reversers!).
- If the Key Generator stub is a secret, encapsulate it in anti-debugging code.
- Evade signature detection using a metamorphic Key Generator stub.
  - Even better, encrypt stub + payload using a good polymorphic encoder.
- Multiple CKP encoders may be applied to a payload.
  - N.B. there's no point in applying the same CKP encoder more than once.

# Conclusion

# Concluding Remarks

- Modern IDSs use dynamic analysis to detect polymorphic malware.
- CKPE prevents the malicious payload from executing within the wrong context (e.g. sandbox, emulator, debugger etc.).
- Introduced 3 new CKP encoders for Metasploit.
- Context-keying is not just for shellcode!
  - Think of PHP code that performs unpacking only when certain data is available at a local database.
- Mitigating CKPE: Making IDSs context-aware.
  - Straightforward for HIDS.
  - Non-trivial for NIDS...

# References


- Snort IPS / IDS
  `http://www.snort.org`
- Network-Level Polymorphic Shellcode Detection Using Emulation, by Polychronakis et al.
  *Journal in Computer Virology, vol. 2, no. 4, pp. 257-274, 2007.*
- Context-keyed Payload Encoding, by |)ruid
  *ToorCon 9, USA, 2007.*
- Advanced Payload Strategies, by R. R. Branco / COSEINC
  *Troopers09, Germany, 2009.*
- Smashing the Stack with Hydra, by Pratap Prabhu et al.
  *DefCon 17, USA, 2009.*
- Metasploit - Penetration Testing Resources
  `http://www.metasploit.com`
- Census CKPE patch for Metasploit Framework
  `http://census-labs.com/media/CKPE-patch`

# QUESTIONS?