

Fuzzing Malware for Fun & Profit. Applying Coverage-Guided Fuzzing to Find Bugs in Modern Malware

Maksim Shudrak

BSides SF 2019

About Me

Interests

Vulnerabilities Hunting
Fuzzing
Reverse-engineering
Malware Analysis
Dynamic Binary Instrumentation

BIO

2018 - present: Senior Offensive Security Researcher
2016: Defended PhD (Vulns Hunting) in Tomsk, Russia
2015-2017: Researcher, IBM Research, Haifa, Israel
2011-2015: Security Researcher, PhD student



Projects

Drltrace - transparent API-calls tracing for malware analysis
<https://github.com/mxmssh/drltrace>
WinHeap Explorer - PoC for heap-based bugs detection in x86 code
<https://github.com/WinHeapExplorer/WinHeap-Explorer>
IDAMetrics - IDA plugin for machine code complexity assessment
<https://github.com/mxmssh/IDAMetrics>



Introduction & Motivation



Why coverage-guided fuzzing ?



Fuzzer overview & architecture



Fuzzer usage & demo



Case Studies. Mirai + vulnerability demo



Case Studies. TinyNuke, Dexter



Discussion, Conclusion

Motivation . Complex Parsers

```
__int32 __cdecl Nspr4Hook::hookerPrWrite(void *fd, const void *buf, __int32 amount)
{
    #if defined WDEBUG1
        WDEBUG1(WDDT_INFO, "Called, amount=%i.", amount);
    #endif

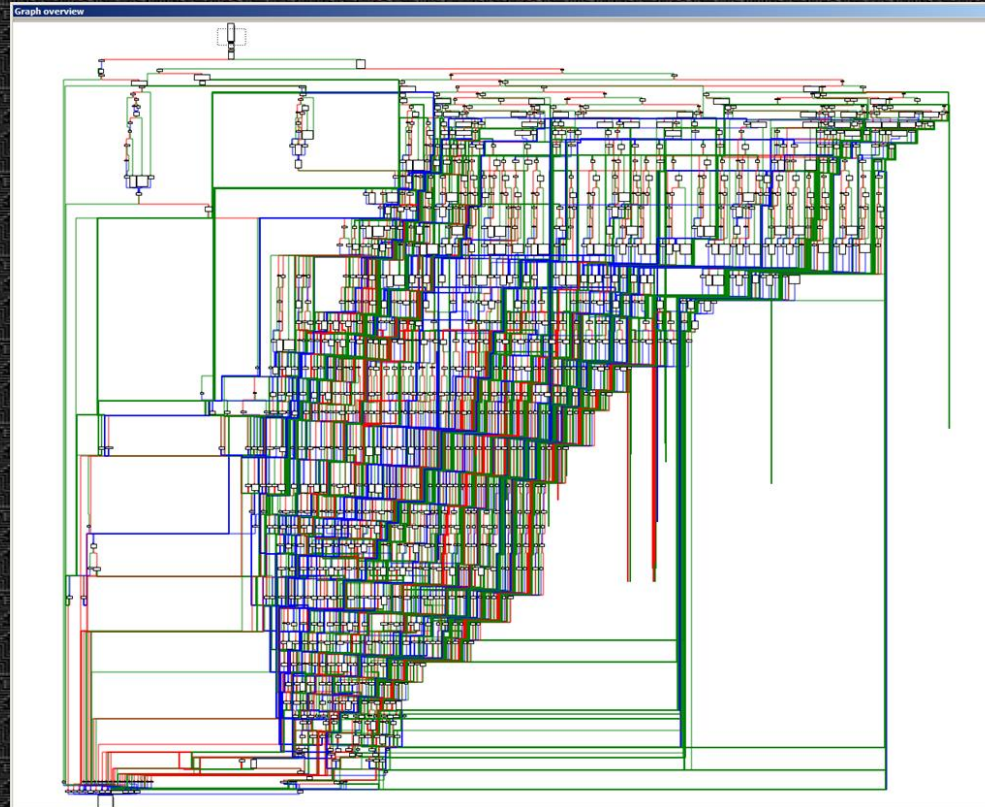
    if(Core::isActive() && buf != NULL && amount > 0)
    {
        /*
         Я просто [REDACTED] писать этот алгоритм. - ☺
        */
        CWA(kernel32, EnterCriticalSection)(&connectionsCs);
        DWORD connectionIndex = connectionFind((PRFILEDESC *)fd);
        if(connectionIndex != (DWORD)-1)
        {
            NSPR4CONNECTION *nc = &connections[connectionIndex];
            #if defined WDEBUG1
                WDEBUG1(WDDT_INFO, "Connection 0x%p founded in table.", fd);
            #endif
        }
    }
}
```


Motivation . Complex Parsers

```
__int32 __cdecl Nspr4Hook::hookerPrWrite(void *fd, const void *buf, __int32 amount)
{
    #if defined WDEBUG1
        WDEBUG1(WDDT_INFO, "Called, amount=%i.", amount);
    #endif

    if(Core::isActive() && buf != NULL && amount > 0)
    {
        /*
         I am so #@$$%^* tired of writing this algorithm.
        */
        CWA(kernel32, EnterCriticalSection) (&connectionsCs);
        DWORD connectionIndex = connectionFind((PRFILEDESC *)fd);
        if(connectionIndex != (DWORD)-1)
        {
            NSPR4CONNECTION *nc = &connections[connectionIndex];
            #if defined WDEBUG1
                WDEBUG1(WDDT_INFO, "Connection 0x%p founded in table.", fd);
            #endif
        }
    }
}
```


Motivation . Complex Parsers



Motivation. Low Code Quality



MalwareTech  @MalwareTechBlog · Jul 7

I finally got the malware sample I was analyzing to give me what I wanted then it immediately crashes because the code was badly written. 🤦



4



2



70



Motivation. It is Fun!



Related Works

- DEF CON 25 Offensive Malware Analysis: Dissecting OSX/FruitFly via a Custom C&C Server by **Patrick Wardle**
- DEF CON 25 Digital Vengeance: Exploiting the Most Notorious C&C Toolkits by **Professor Plum**
- Targeted attacks: From being a victim to counter attacking by **Andrzej Dereszowski** (SIGNAL 11)
- **Malware fuzzing:**
 - Rasthofer, S., Arzt, S., Triller, S. and Pradel, M., 2017, May. Making malory behave maliciously: Targeted fuzzing of android execution environments. In Software Engineering (ICSE), 2017 IEEE/ACM 39th International Conference on (pp. 300-311). IEEE.
 - F. Peng, Z. Deng, X. Zhang, D. Xu, Z. Lin, and Z. Su. X-force: Force executing binary programs for security applications. In Proceedings of the 2014 USENIX Security Symposium, San Diego, CA (August 2014), 2014

Legal Issues

- Hacking-back is mostly illegal
 - Attack attribution is very hard and might lead to wrong conclusions
 - Hard to identify scopes of attack
 - Check out last year DEF CON Professor Plum's presentation for more details:
 - <https://www.youtube.com/watch?v=fPhkmAdWH-I>
- BUT no one can prohibit us to search for bugs in malware

Possible Benefits. Local Deny of Service (agent)



The image is a screenshot of a Wired news article. At the top, the Wired logo is on the left, and the article title "How an Accidental 'Kill Switch' Slowed Friday's Massive Ransomware Attack" is on the right. Below the title is a navigation bar with categories: BUSINESS, GEAR, IDEAS, SCIENCE, SECURITY (highlighted), and TRANSPORTATION. The article byline reads "LILY HAY NEWMAN SECURITY 05.13.17 03:27 PM". The main headline is "HOW AN ACCIDENTAL 'KILL SWITCH' SLOWED FRIDAY'S MASSIVE RANSOMWARE ATTACK". On the left side, there is a "SHARE" section with a Facebook share button showing 2162 shares and a Twitter share button. On the right side, there is a partial view of another article titled "MOS" with a small image of a person's hands.

Wired

How an Accidental 'Kill Switch' Slowed Friday's Massive Ransomware Attack

BUSINESS GEAR IDEAS SCIENCE **SECURITY** TRANSPORTATION

LILY HAY NEWMAN SECURITY 05.13.17 03:27 PM

HOW AN ACCIDENTAL 'KILL SWITCH' SLOWED FRIDAY'S MASSIVE RANSOMWARE ATTACK

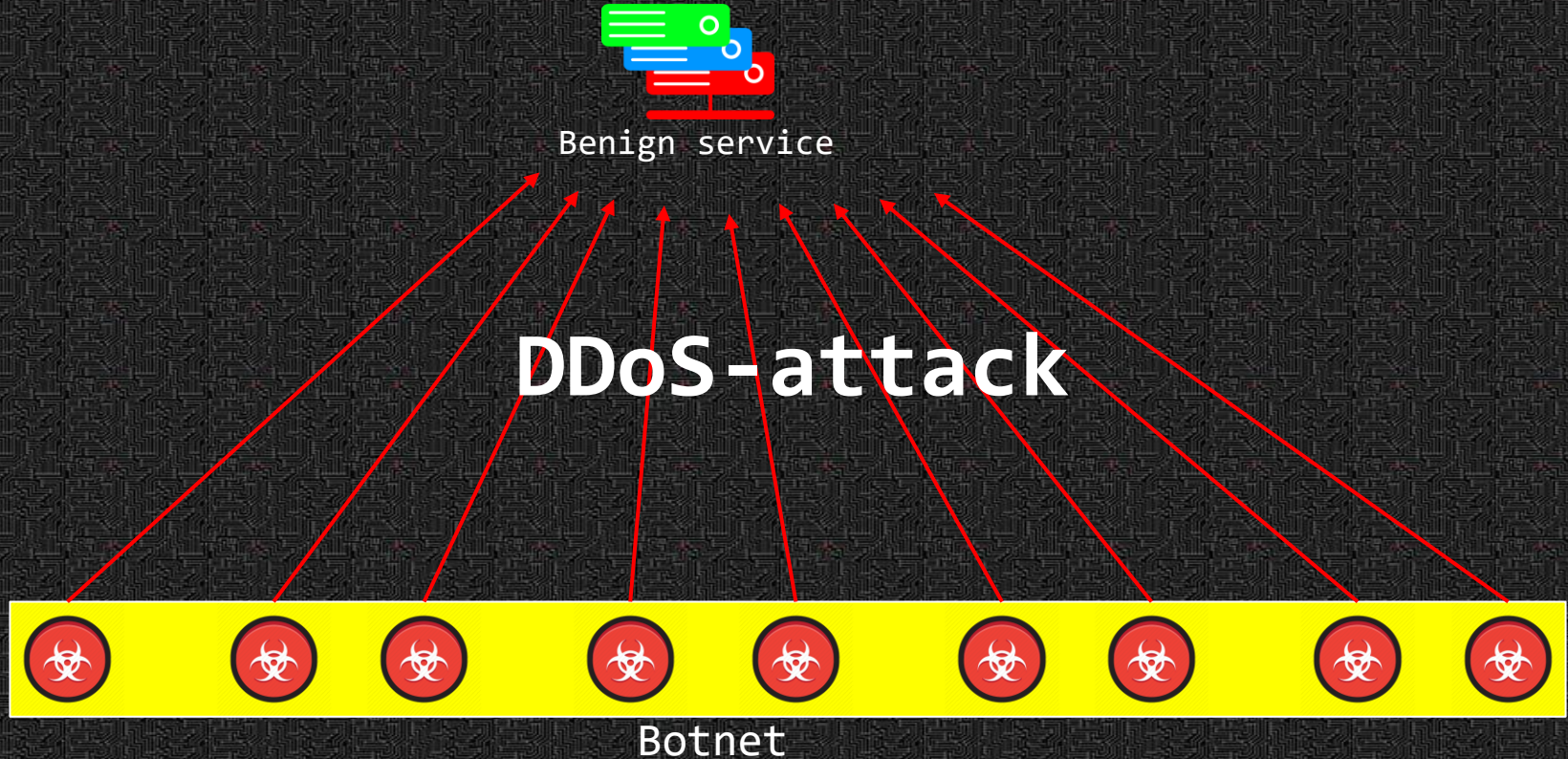
SHARE

f SHARE 2162

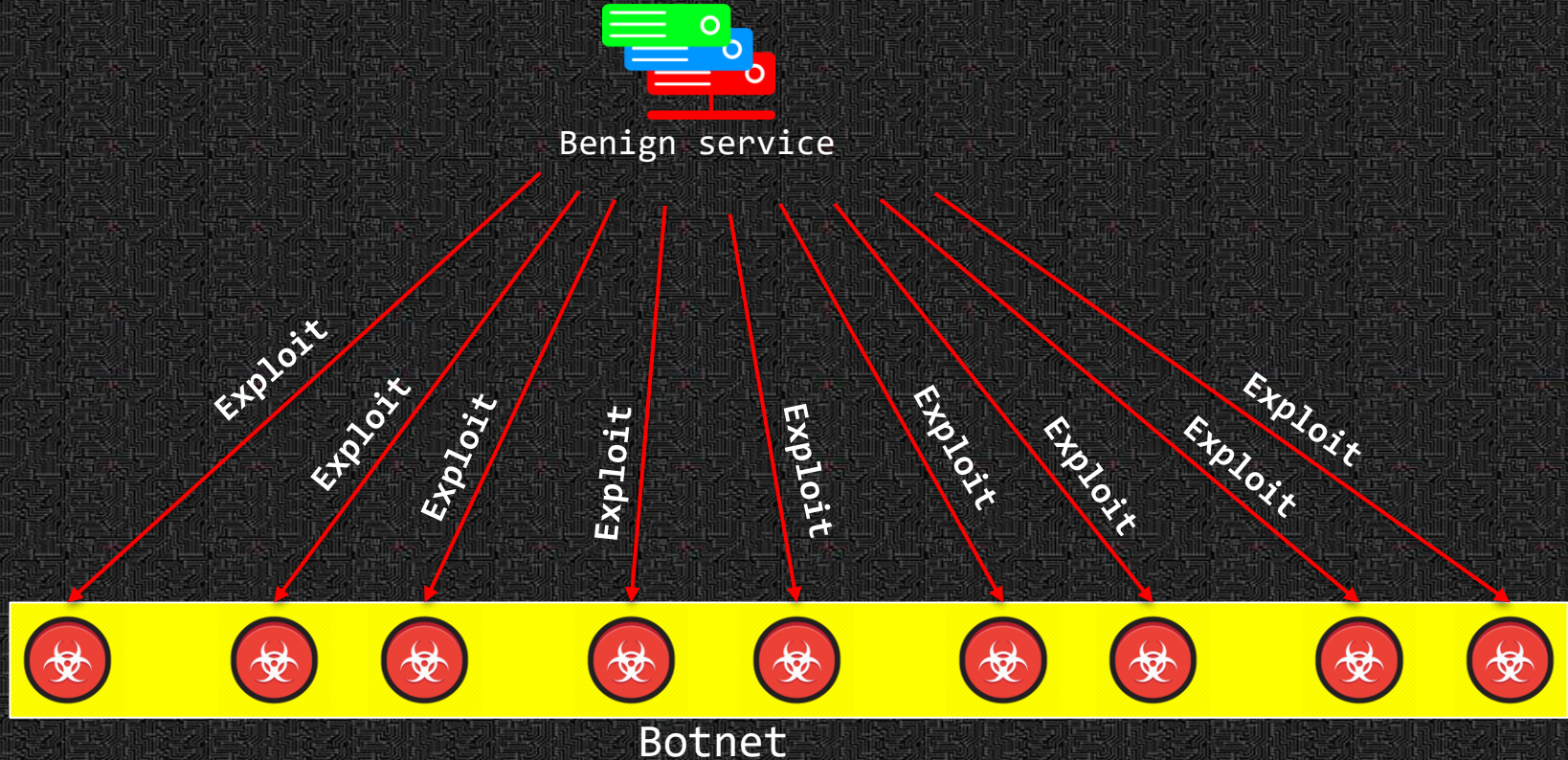
🐦 TWEET

MOS

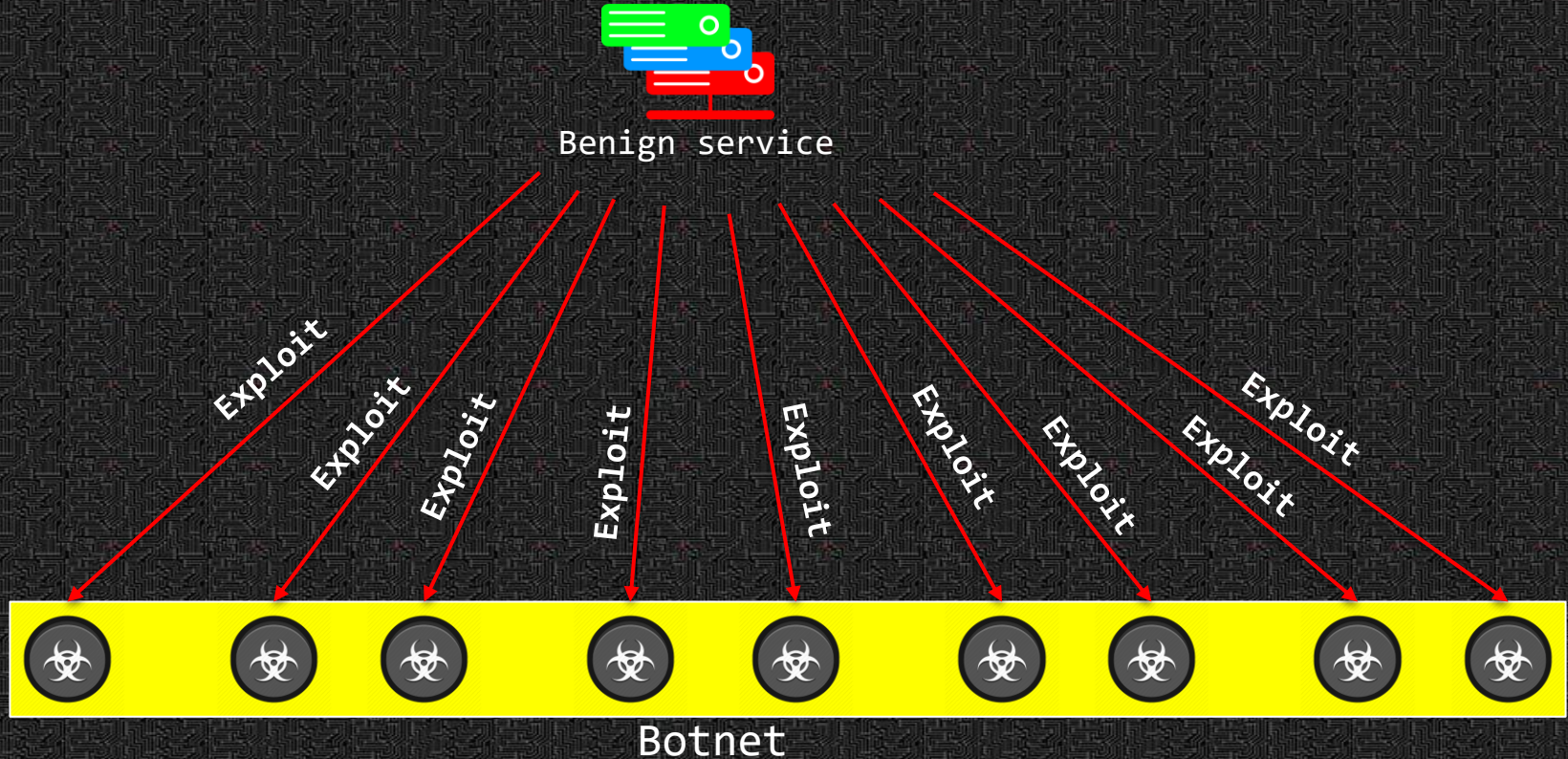
Possible Benefits. Remote Deny of Service (agent)



Possible Benefits. Remote Deny of Service (agent)



Possible Benefits. Remote Deny of Service (agent)



Possible Benefits. Remote Code Execution (agent)

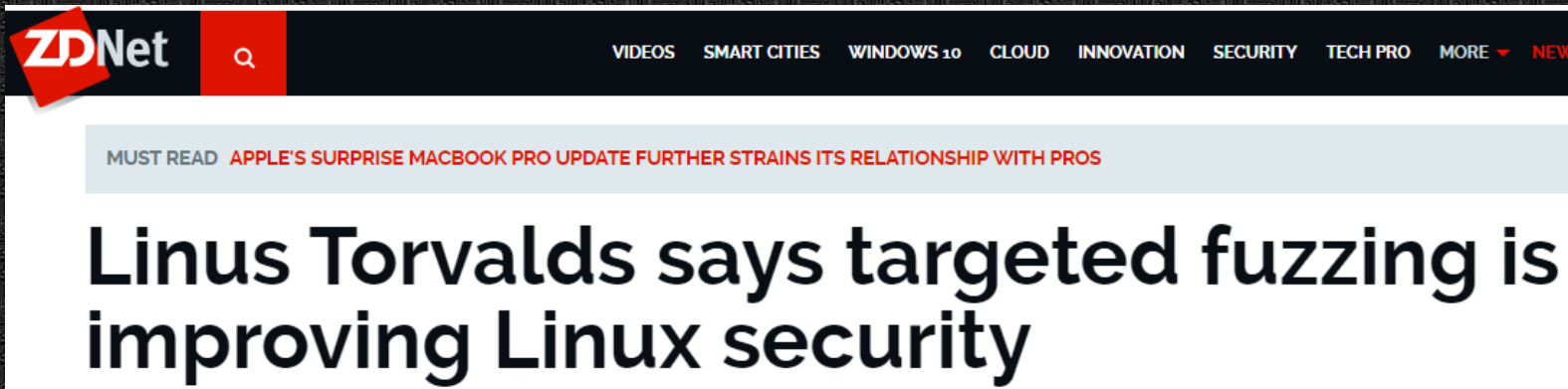
1. Take control over botnet or shutdown botnet
2. Track down botnet owners
3. ?????
4. PROFIT

Possible Benefits. Remote Code Execution in C&C



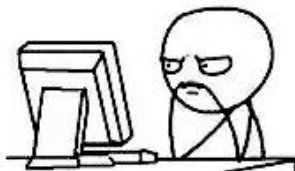
Fuzzing

- Nowadays, fuzzing is a state-of-the-art approach to find bugs in modern applications
- Fuzzing is a part of SDLC
- Fuzzing is very important for applications & OS security



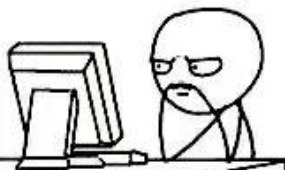
Fuzzing

```
afl-fuzz -i in -o out target @@
```

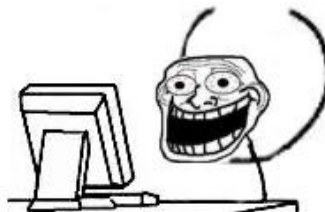


```
overall results  
cycles done : 0  
total paths : 146  
uniq crashes : 0  
uniq hangs : 0
```

3 hours later



```
overall results  
cycles done : 0  
total paths : 146  
uniq crashes : 1  
uniq hangs : 0
```



What is Coverage-Guided Fuzzing ?

```
AIJsonError ParseValue(AIJson *json, char **str, AIJsonValue *value)
```

```
{  
    AIJsonError err;  
    switch(**str)  
    {  
        case CHAR_STR_OPEN_CLOSE:  
        {  
            char *data;  
            value->type = AI_JSON_STRING;  
            err = ParseString(json, str, &data);  
            if(err != AI_JSON_E_OK)  
                return err;  
            value->data.string = data;  
            break;  
        }  
        case CHAR_OBJECT_OPEN:  
        {  
            value->type = AI_JSON_OBJECT;  
            value->data.object = JsonListCreate(json);  
            if(!value->data.object)  
                return AI_JSON_E_ALLOC;  
            err = ParseObject(json, str, value->data.array);  
            if(err != AI_JSON_E_OK)  
                return err;  
            break;  
        }  
        case CHAR_ARRAY_OPEN:  
        {  
            value->type = AI_JSON_ARRAY;  
            value->data.array = JsonListCreate(json);  
            if(!value->data.array)  
                return AI_JSON_E_ALLOC;  
            err = ParseArray(json, str, value->data.array);  
            if(err != AI_JSON_E_OK)  
                return err;  
            break;  
        }  
    }  
}
```

american fuzzy lop 0.47b (readpng)

process timing		overall results	
run time	: 0 days, 0 hrs, 4 min, 43 sec	cycles done	: 0
last new path	: 0 days, 0 hrs, 0 min, 26 sec	total paths	: 195
last uniq crash	: none seen yet	uniq crashes	: 0
last uniq hang	: 0 days, 0 hrs, 1 min, 51 sec	uniq hangs	: 1
cycle progress		map coverage	
now processing	: 38 (19.49%)	map density	: 1217 (7.43%)
paths timed out	: 0 (0.00%)	count coverage	: 2.55 bits/tuple
stage progress		findings in depth	
now trying	: interest 32/8	favored paths	: 128 (65.64%)
stage execs	: 0/9990 (0.00%)	new edges on	: 85 (43.59%)
total execs	: 654k	total crashes	: 0 (0 unique)
exec speed	: 2306/sec	total hangs	: 1 (1 unique)
fuzzing strategy yields		path geometry	
bit flips	: 88/14.4k, 6/14.4k, 6/14.4k	levels	: 3
byte flips	: 0/1804, 0/1786, 1/1750	pending	: 178
arithmetics	: 31/126k, 3/45.6k, 1/17.8k	pend fav	: 114
known ints	: 1/15.8k, 4/65.8k, 6/78.2k	imported	: 0
havoc	: 34/254k, 0/0	variable	: 0
trim	: 2876 B/931 (61.45% gain)	latent	: 0

What is Coverage-Guided Fuzzing ?

```
AiJsonError ParseValue(AiJson *json, char **str, AiJsonValue *value)
```

```
{  
    AiJsonError err;  
    switch(**str)  
    {  
        case CHAR_STR_OPEN_CLOSE:  
        {  
            char *data;  
            value->type = AI_JSON_STRING;  
            err = ParseString(json, str, &data);  
            if(err != AI_JSON_E_OK)  
                return err;  
            value->data.string = data;  
            break;  
        }  
        case CHAR_OBJECT_OPEN:  
        {  
            value->type = AI_JSON_OBJECT;  
            value->data.object = JsonListCreate(json);  
            if(!value->data.object)  
                return AI_JSON_E_ALLOC;  
            err = ParseObject(json, str, value->data.array);  
            if(err != AI_JSON_E_OK)  
                return err;  
            break;  
        }  
        case CHAR_ARRAY_OPEN:  
        {  
            value->type = AI_JSON_ARRAY;  
            value->data.array = JsonListCreate(json);  
            if(!value->data.array)  
                return AI_JSON_E_ALLOC;  
            err = ParseArray(json, str, value->data.array);  
            if(err != AI_JSON_E_OK)  
                return err;  
            break;  
        }  
    }  
}
```

Record new path

american fuzzy lop 0.47b (readpng)

process timing		overall results	
run time	: 0 days, 0 hrs, 4 min, 43 sec	cycles done	: 0
last new path	: 0 days, 0 hrs, 0 min, 26 sec	total paths	: 195
last uniq crash	: none seen yet	uniq crashes	: 0
last uniq hang	: 0 days, 0 hrs, 1 min, 51 sec	uniq hangs	: 1
cycle progress		map coverage	
now processing	: 38 (19.49%)	map density	: 1217 (7.43%)
paths timed out	: 0 (0.00%)	count coverage	: 2.55 bits/tuple
stage progress		findings in depth	
now trying	: interest 32/8	favored paths	: 128 (65.64%)
stage execs	: 0/9990 (0.00%)	new edges on	: 85 (43.59%)
total execs	: 654k	total crashes	: 0 (0 unique)
exec speed	: 2306/sec	total hangs	: 1 (1 unique)
fuzzing strategy yields		path geometry	
bit flips	: 88/14.4k, 6/14.4k, 6/14.4k	levels	: 3
byte flips	: 0/1804, 0/1786, 1/1750	pending	: 178
arithmetics	: 31/126k, 3/45.6k, 1/17.8k	pend fav	: 114
known ints	: 1/15.8k, 4/65.8k, 6/78.2k	imported	: 0
havoc	: 34/254k, 0/0	variable	: 0
trim	: 2876 B/931 (61.45% gain)	latent	: 0

What is Coverage-Guided Fuzzing ?

```
AIJsonError ParseValue(AIJson *json, char **str, AIJsonValue *value)
```

```
{  
    AIJsonError err;  
    switch(**str)  
    {  
        case CHAR_STR_OPEN_CLOSE:  
        {  
            char *data;  
            value->type = AI_JSON_STRING;  
            err = ParseString(json, str, &data);  
            if(err != AI_JSON_E_OK)  
                return err;  
            value->data.string = data;  
            break;  
        }  
        case CHAR_OBJECT_OPEN:  
        {  
            value->type = AI_JSON_OBJECT;  
            value->data.object = JsonListCreate(json);  
            if(!value->data.object)  
                return AI_JSON_E_ALLOC;  
            err = ParseObject(json, str, value->data.array);  
            if(err != AI_JSON_E_OK)  
                return err;  
            break;  
        }  
        case CHAR_ARRAY_OPEN:  
        {  
            value->type = AI_JSON_ARRAY;  
            value->data.array = JsonListCreate(json);  
            if(!value->data.array)  
                return AI_JSON_E_ALLOC;  
            err = ParseArray(json, str, value->data.array);  
            if(err != AI_JSON_E_OK)  
                return err;  
            break;  
        }  
    }  
}
```

Record new path

american fuzzy lop 0.47b (readpng)

process timing		overall results	
run time	: 0 days, 0 hrs, 4 min, 43 sec	cycles done	: 0
last new path	: 0 days, 0 hrs, 0 min, 26 sec	total paths	: 195
last uniq crash	: none seen yet	uniq crashes	: 0
last uniq hang	: 0 days, 0 hrs, 1 min, 51 sec	uniq hangs	: 1
cycle progress		map coverage	
now processing	: 38 (19.49%)	map density	: 1217 (7.43%)
paths timed out	: 0 (0.00%)	count coverage	: 2.55 bits/tuple
stage progress		findings in depth	
now trying	: interest 32/8	favored paths	: 128 (65.64%)
stage execs	: 0/9990 (0.00%)	new edges on	: 85 (43.59%)
total execs	: 654k	total crashes	: 0 (0 unique)
exec speed	: 2306/sec	total hangs	: 1 (1 unique)
fuzzing strategy yields		path geometry	
bit flips	: 88/14.4k, 6/14.4k, 6/14.4k	levels	: 3
byte flips	: 0/1804, 0/1786, 1/1750	pending	: 178
arithmetics	: 31/126k, 3/45.6k, 1/17.8k	pend fav	: 114
known ints	: 1/15.8k, 4/65.8k, 6/78.2k	imported	: 0
havoc	: 34/254k, 0/0	variable	: 0
trim	: 2876 B/931 (61.45% gain)	latent	: 0

What is Coverage-Guided Fuzzing ?

```
AiJsonError ParseValue(AiJson *json, char **str, AiJsonValue *value)
```

```
{  
    AiJsonError err;  
    switch(**str)  
    {  
        case CHAR_STR_OPEN_CLOSE:  
        {  
            char *data;  
            value->type = AI_JSON_STRING;  
            err = ParseString(json, str, &data);  
            if(err != AI_JSON_E_OK)  
                return err;  
            value->data.string = data;  
            break;  
        }  
        case CHAR_OBJECT_OPEN:  
        {  
            value->type = AI_JSON_OBJECT;  
            value->data.object = JsonListCreate(json);  
            if(!value->data.object)  
                return AI_JSON_E_ALLOC;  
            err = ParseObject(json, str, value->data.array);  
            if(err != AI_JSON_E_OK)  
                return err;  
            break;  
        }  
        case CHAR_ARRAY_OPEN:  
        {  
            value->type = AI_JSON_ARRAY;  
            value->data.array = JsonListCreate(json);  
            if(!value->data.array)  
                return AI_JSON_E_ALLOC;  
            err = ParseArray(json, str, value->data.array);  
            if(err != AI_JSON_E_OK)  
                return err;  
            break;  
        }  
    }  
}
```

Record new path

american fuzzy lop 0.47b (readpng)

process timing		overall results	
run time	: 0 days, 0 hrs, 4 min, 43 sec	cycles done	: 0
last new path	: 0 days, 0 hrs, 0 min, 26 sec	total paths	: 195
last uniq crash	: none seen yet	uniq crashes	: 0
last uniq hang	: 0 days, 0 hrs, 1 min, 51 sec	uniq hangs	: 1
cycle progress		map coverage	
now processing	: 38 (19.49%)	map density	: 1217 (7.43%)
paths timed out	: 0 (0.00%)	count coverage	: 2.55 bits/tuple
stage progress		findings in depth	
now trying	: interest 32/8	favored paths	: 128 (65.64%)
stage execs	: 0/9990 (0.00%)	new edges on	: 85 (43.59%)
total execs	: 654k	total crashes	: 0 (0 unique)
exec speed	: 2306/sec	total hangs	: 1 (1 unique)
fuzzing strategy yields		path geometry	
bit flips	: 88/14.4k, 6/14.4k, 6/14.4k	levels	: 3
byte flips	: 0/1804, 0/1786, 1/1750	pending	: 178
arithmetics	: 31/126k, 3/45.6k, 1/17.8k	pend fav	: 114
known ints	: 1/15.8k, 4/65.8k, 6/78.2k	imported	: 0
havoc	: 34/254k, 0/0	variable	: 0
trim	: 2876 B/931 (61.45% gain)	latent	: 0

Why Coverage-Guided Fuzzing ?

~minutes for AFL and thousand years for dump fuzzer

```
18  if (userBuffer[0] == 'P'){
19      if (userBuffer[1] == 'w'){
20          if (userBuffer[2] == 'n'){
21              if (userBuffer[3] == 'T'){
22                  if (userBuffer[4] == 'o'){
23                      if (userBuffer[5] == 'w'){
24                          if (userBuffer[6] == 'n'){
25                              /* hell yeah */
26                              ((VOID(*)())0x0)();
27                          }
28                      }
29                  }
30              }
31          }
32      }
33  }
```


State-of-the-art Coverage-Guided Fuzzers

- AFL
 - <http://lcamtuf.coredump.cx/afl/>
- Libfuzzer
 - <https://llvm.org/docs/LibFuzzer.html>
- AFL's forks
 - kAFL – AFL for kernel-level fuzzing
 - WinAFL – AFL fork for Windows binaries fuzzing
 - and many others:
https://github.com/mirrorer/afl/blob/master/docs/sister_projects.txt

AFL Source Code Instrumentation Approach

- Custom gcc (afl-gcc) compiler is used to inject instrumentation routines for each basic block
- Main routine after instrumentation looks like this:

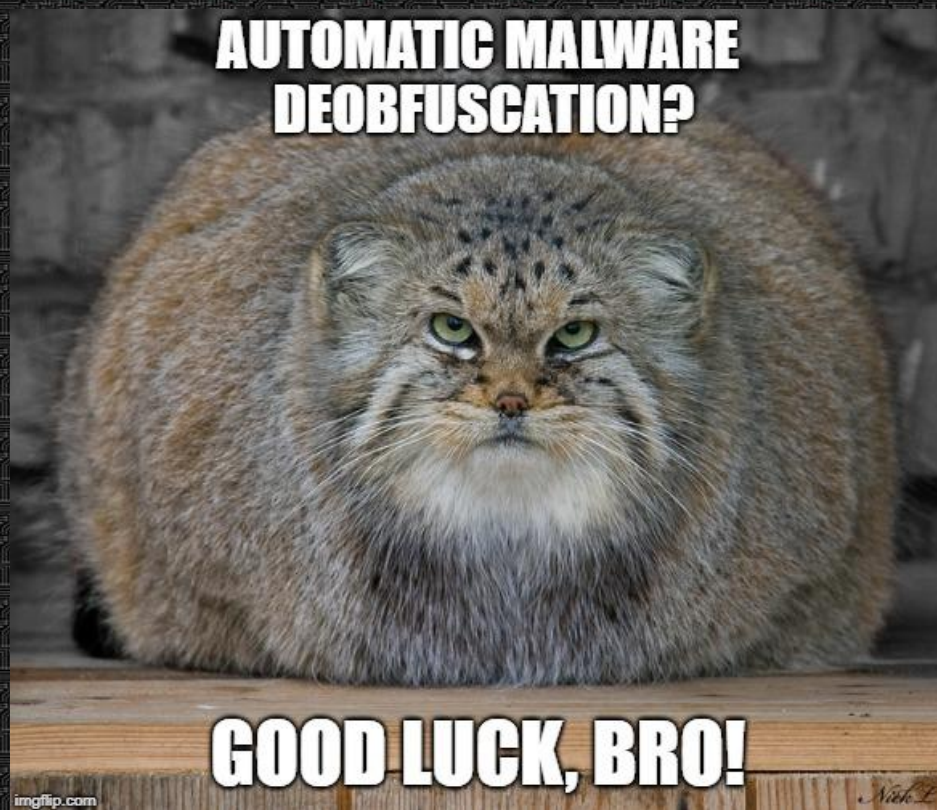
```
0x4009d0 <main>      lea    -0x98(%rsp),%rsp
0x4009d8 <main+8>     mov     %rdx,(%rsp)
0x4009dc <main+12>    mov     %rcx,0x8(%rsp)
0x4009e1 <main+17>    mov     %rax,0x10(%rsp)
0x4009e6 <main+22>    mov     $0x2cf4,%rcx
0x4009ed <main+29>    callq   0x4034c0 < afl maybe log>
0x4009f2 <main+34>    mov     0x10(%rsp),%rax
```


Challenge I. Source Code

No Source Code

```
35     Size = sizeof(UserAgent);
36     _memset(UserAgent,0x00,Size);
37     ObtainUserAgentString(0,UserAgent,&Size);
38     if(UserAgent[0]==0x00) { lstrcpy(UserAgent,"Mozilla/4.0(compatible; MSIE 7.0b; Windows
39
40     while((hOpen = InternetOpen(UserAgent,INTERNET_OPEN_TYPE_PRECONFIG,NULL,NULL,0)) == NU
41
42     x = 0;
43     ValidIndex = FALSE;
44     AlreadyConnected = FALSE;
45     PresistInfo = FALSE;
46     while(1) { //Start enumerating URLs till you find working one
47
48     if(Urls[x]==0x00) {
49         x = 0;
50         Sleep(ConnectInterval);
51     }
52 }
```


Challenge II. Obfuscation



Challenge III. Encryption

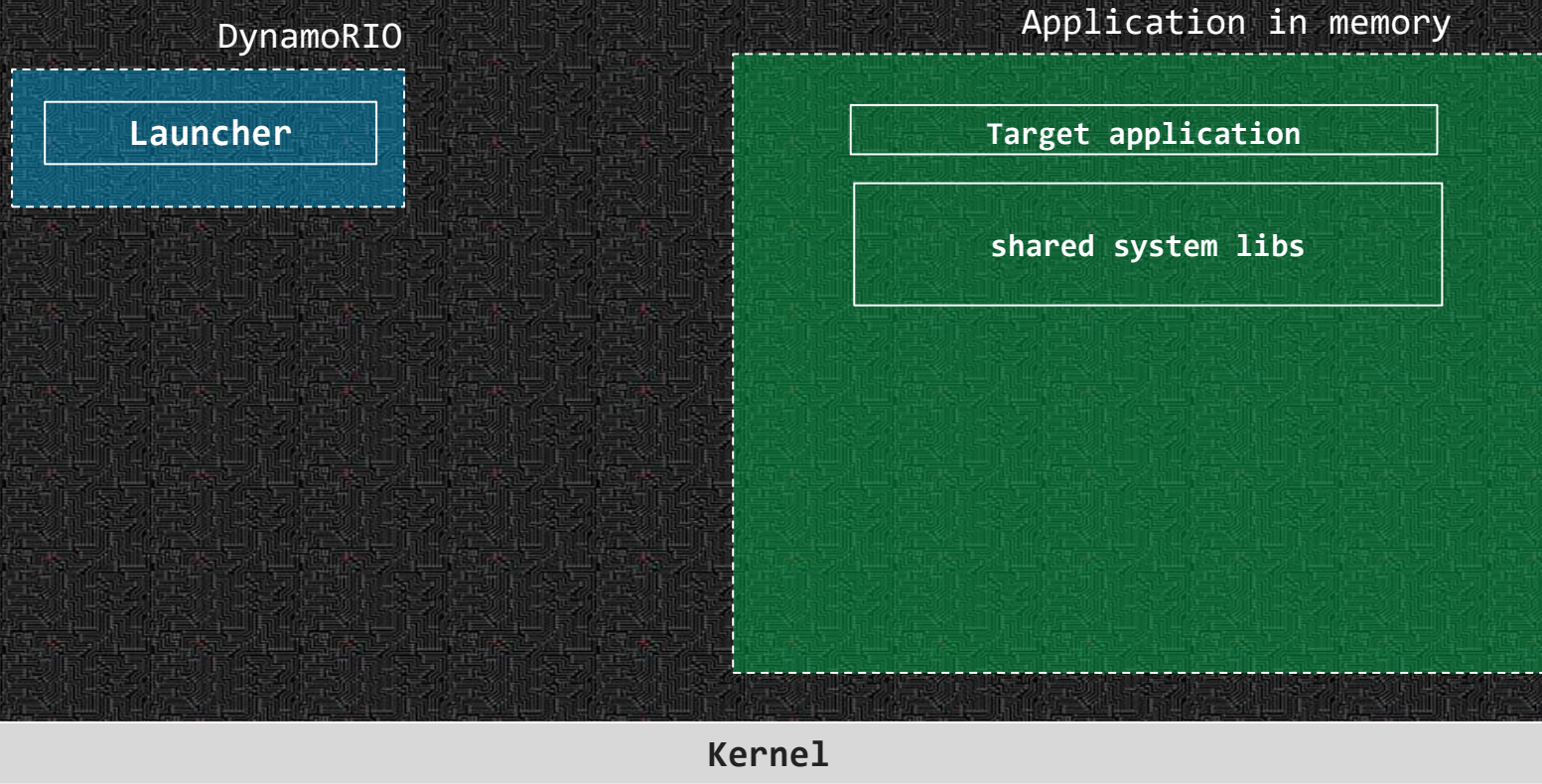
- Most C&C channels are encrypted
- We need to encrypt our test case the same way as malware to be able to find bugs
- By default, AFL doesn't support encryption, checksums and crypto signatures generation
 - There is a post processor library to deal with that

WinAFL

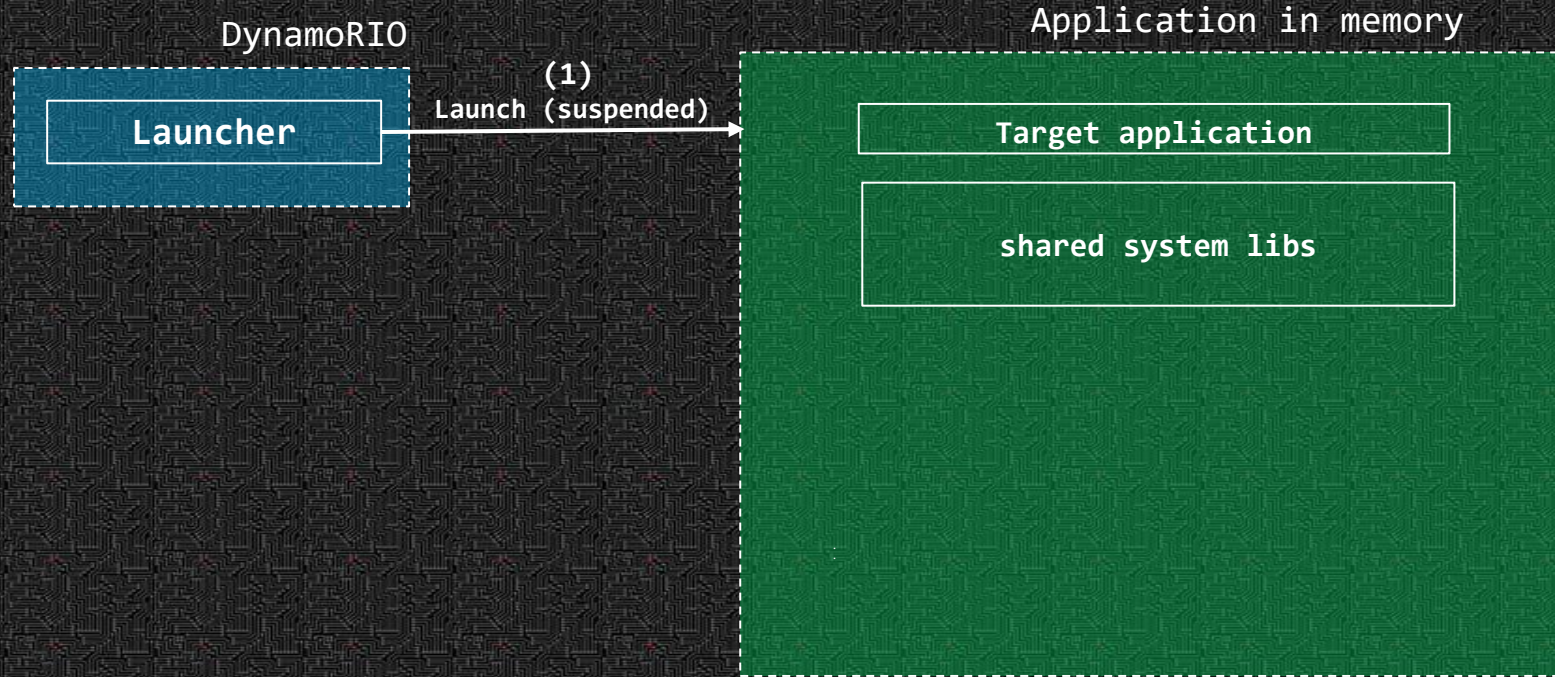
- WinAFL is a port of AFL for Windows. Rely on DynamoRIO dynamic binary instrumentation framework.
 - No need for source code access
 - Open-source
 - Fast-enough to use for coverage-guided fuzzing
- <https://github.com/ivanfratric/winafl>
<https://github.com/DynamoRIO/dynamorio>

Dynamic Binary Instrumentation (DBI) is a technique of analyzing the behavior of a binary application at runtime through the injection of instrumentation code.

How Does DynamoRIO Work ? (10000 foot view)

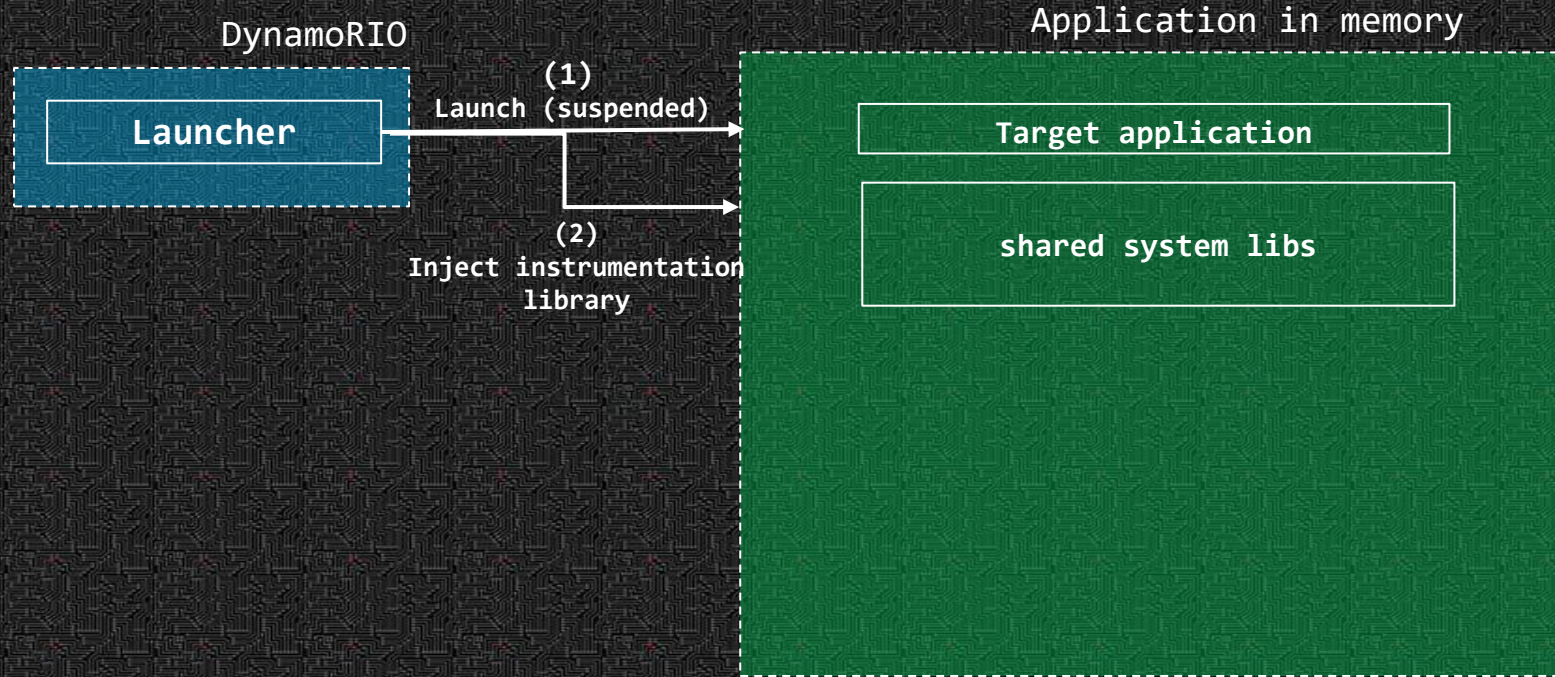


How Does DynamoRIO Work ? (10000 foot view)

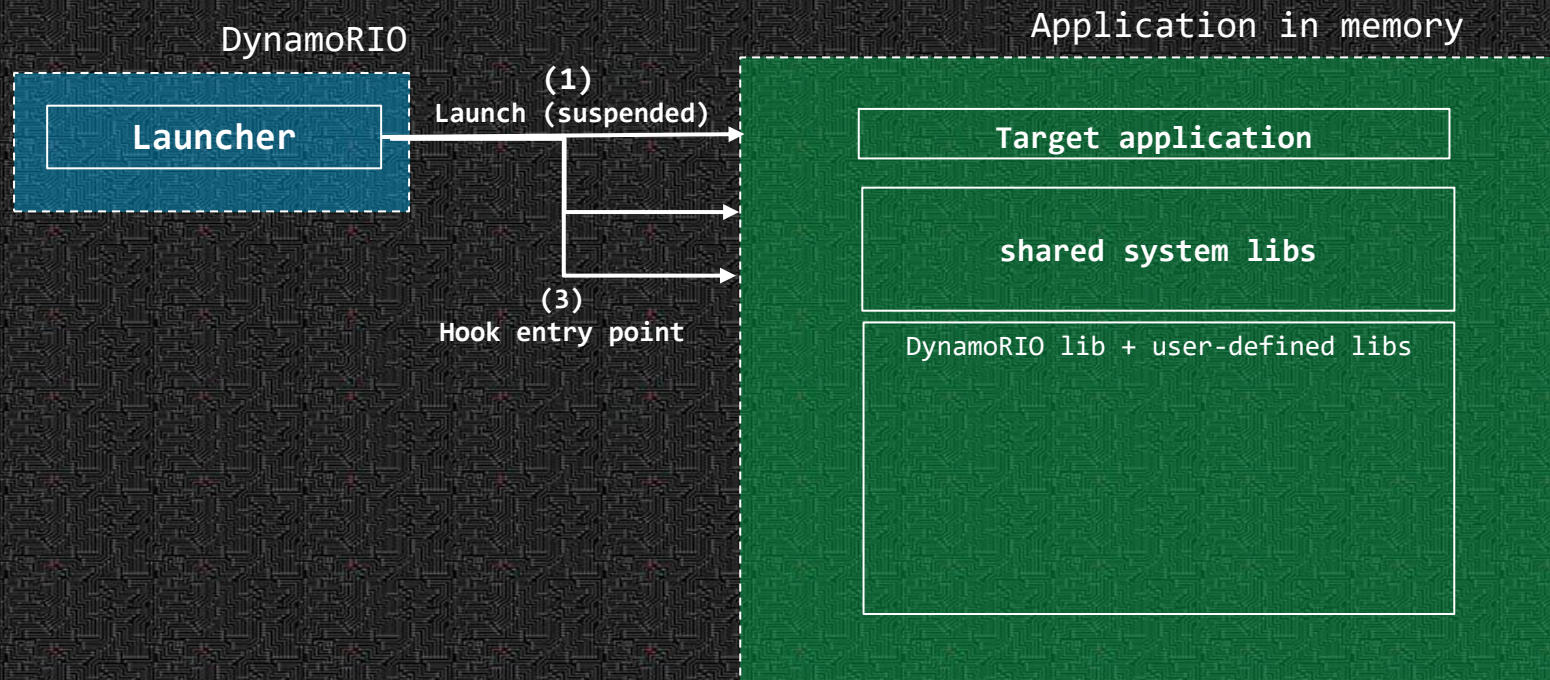


Kernel

How Does DynamoRIO Work ? (10000 foot view)

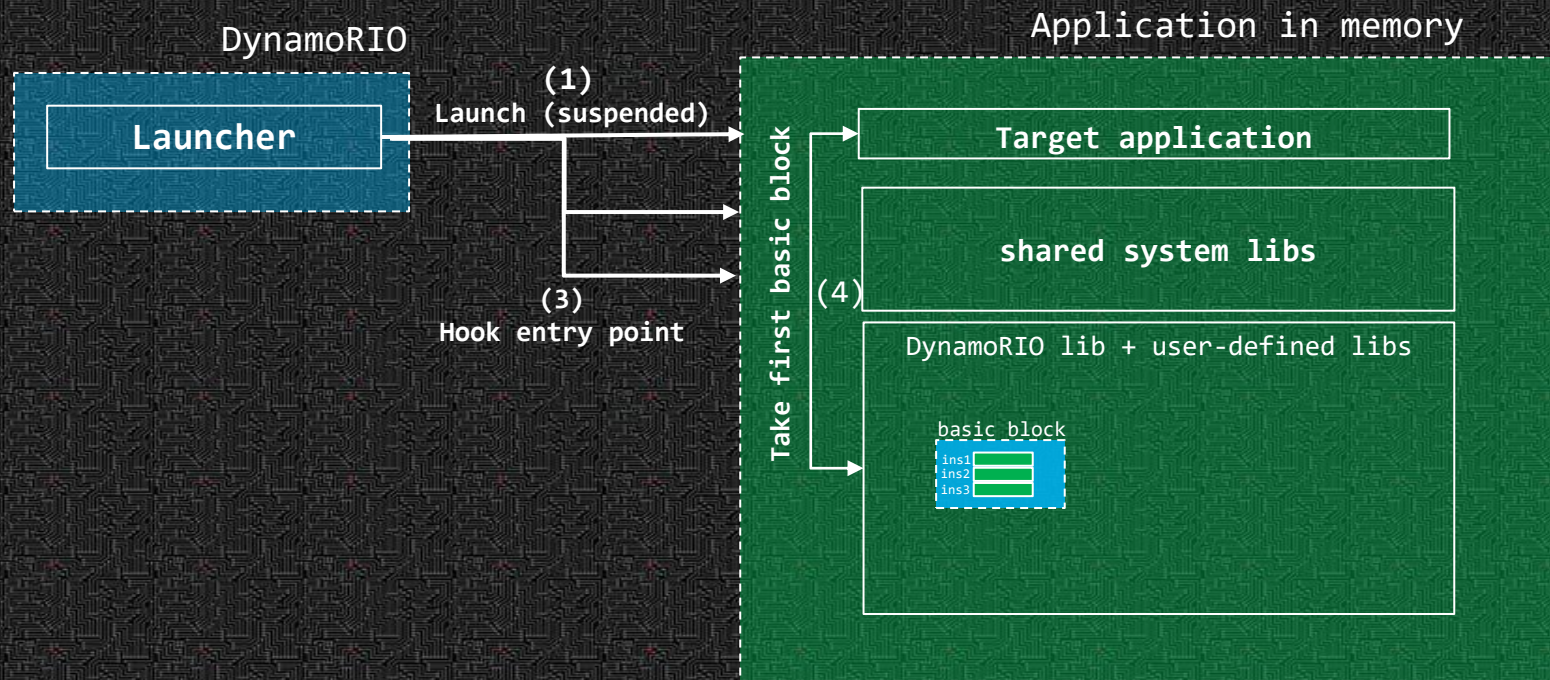


How Does DynamoRIO Work ? (10000 foot view)

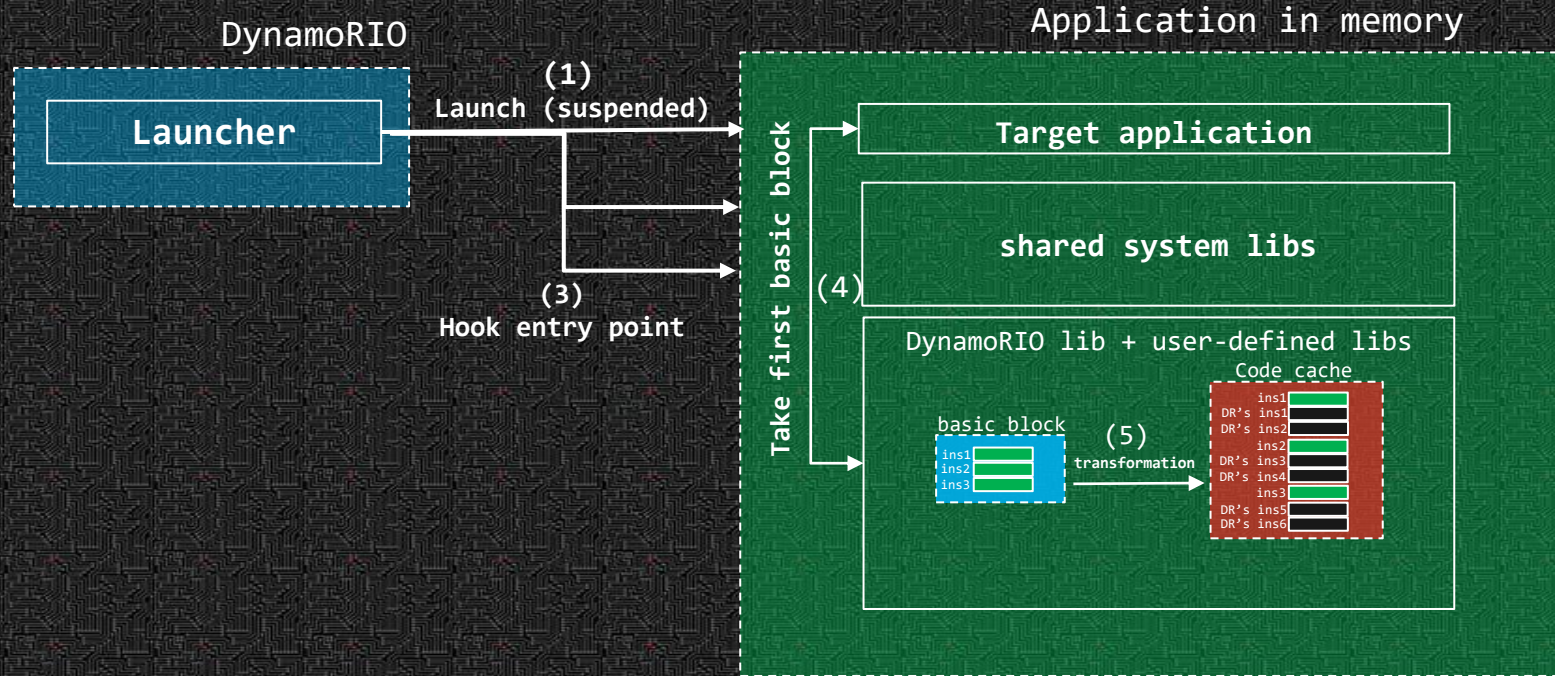


Kernel

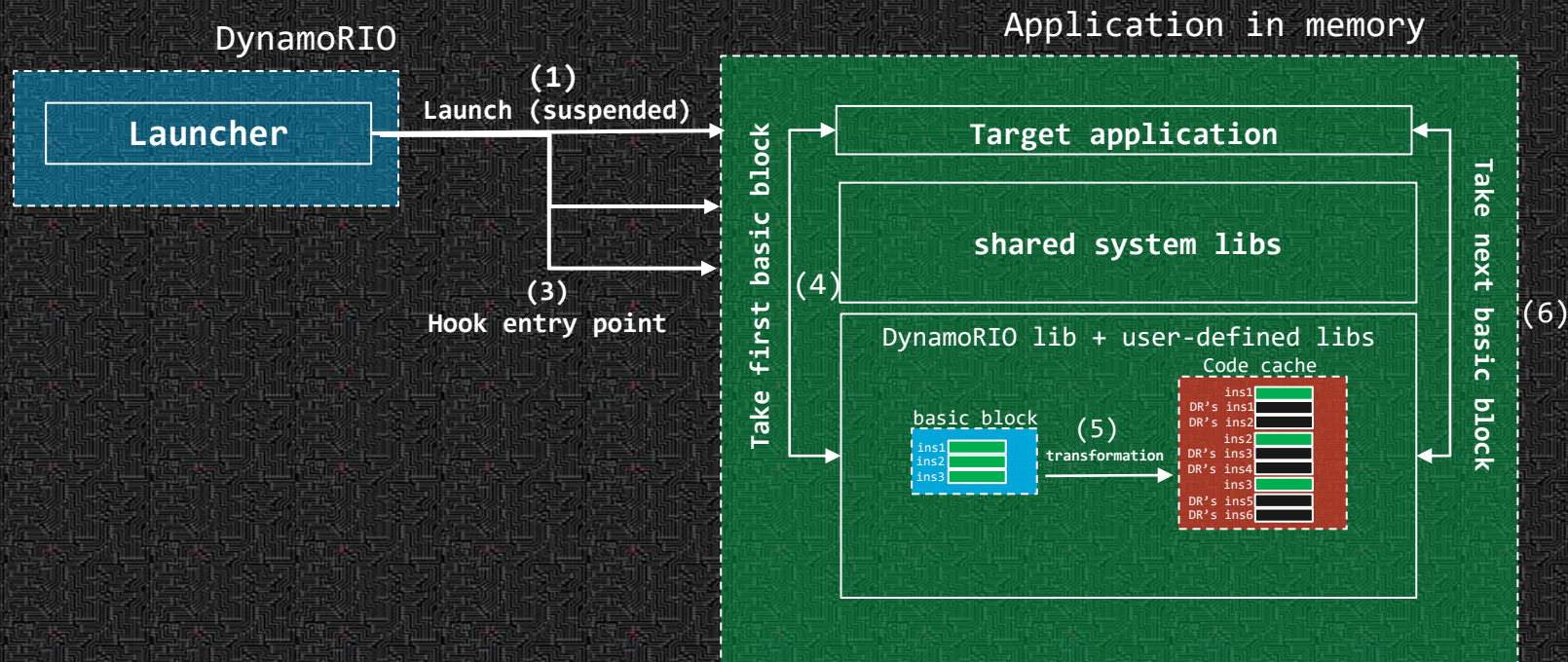
How Does DynamoRIO Work ? (10000 foot view)



How Does DynamoRIO Work ? (10000 foot view)



How Does DynamoRIO Work ? (10000 foot view)



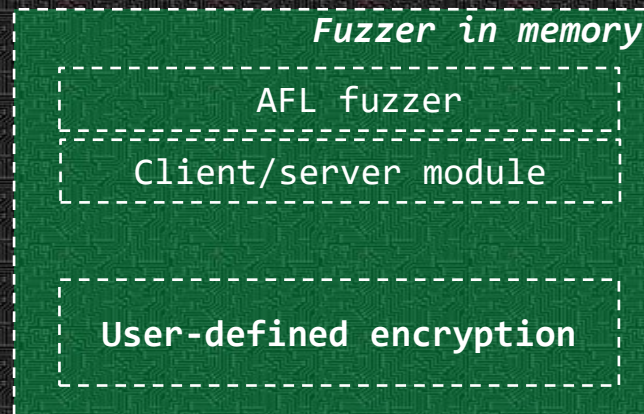
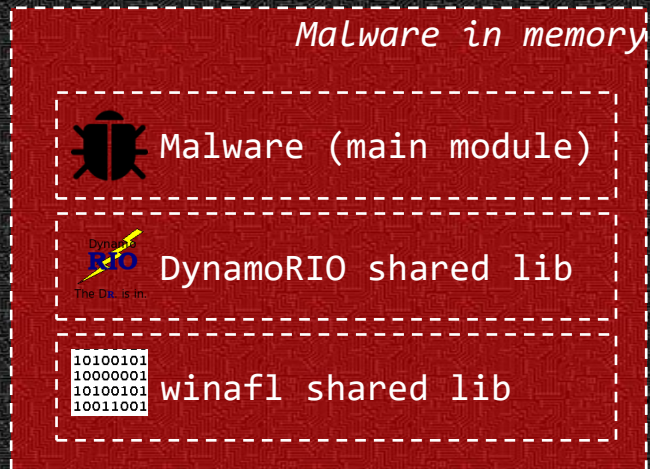
Challenges

- Lack of source code
- Obfuscation
- Encryption

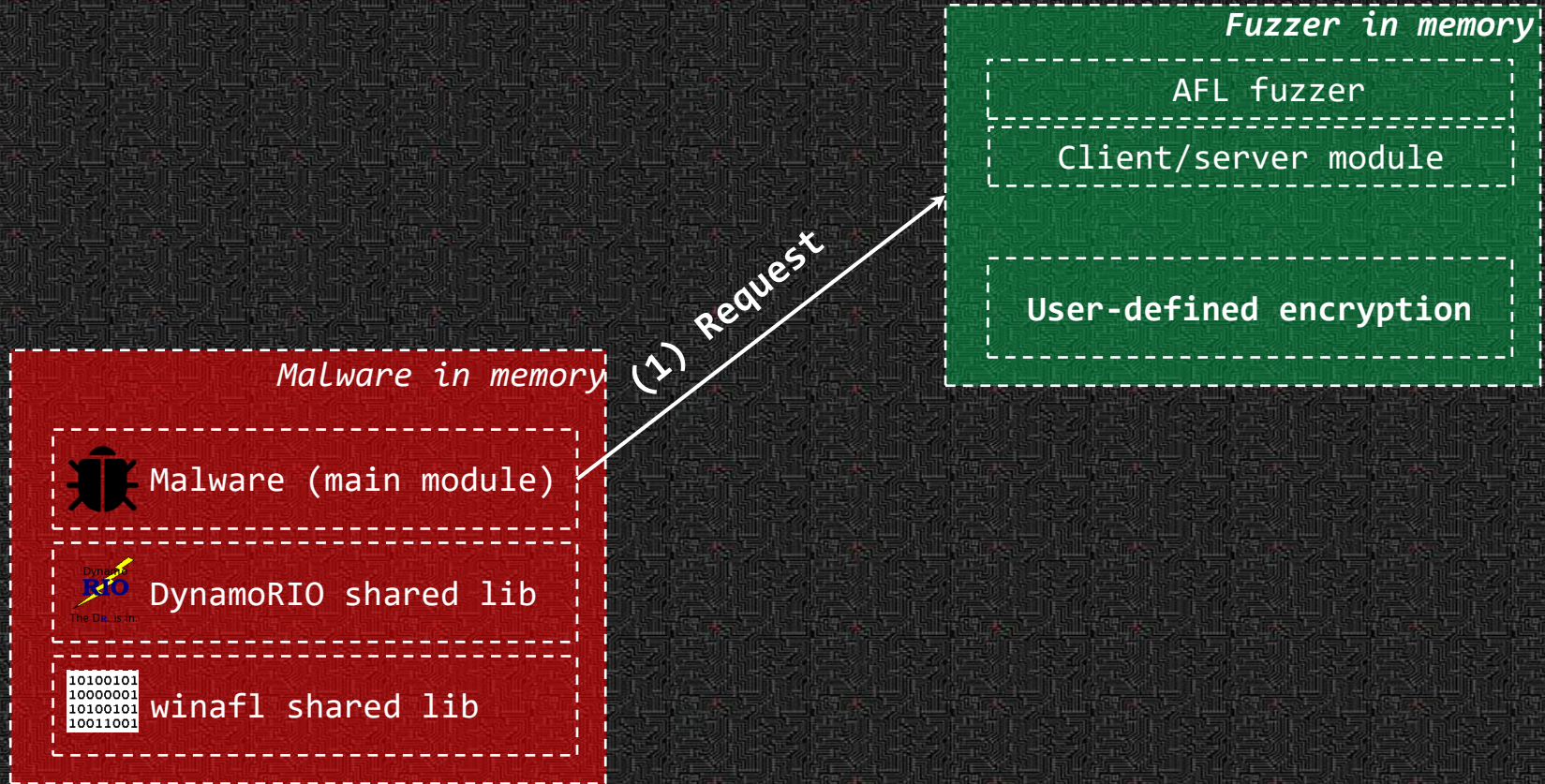
Challenges

- ~~Lack of source code~~ – WinAFL + DynamoRIO
- WinAFL supports only file-based fuzzing
- Obfuscation
- Encryption

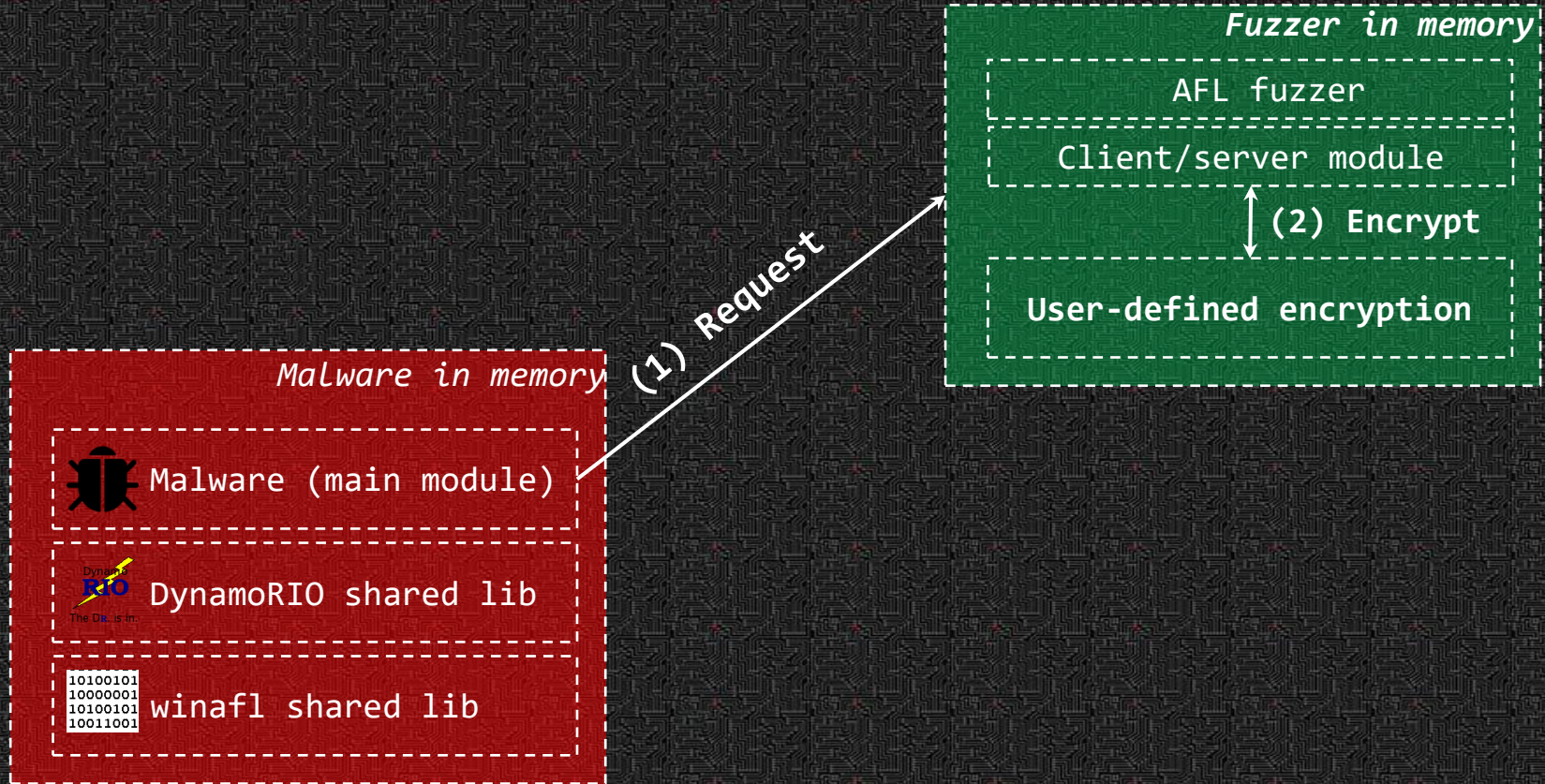
WinAFL patch (netAFL)



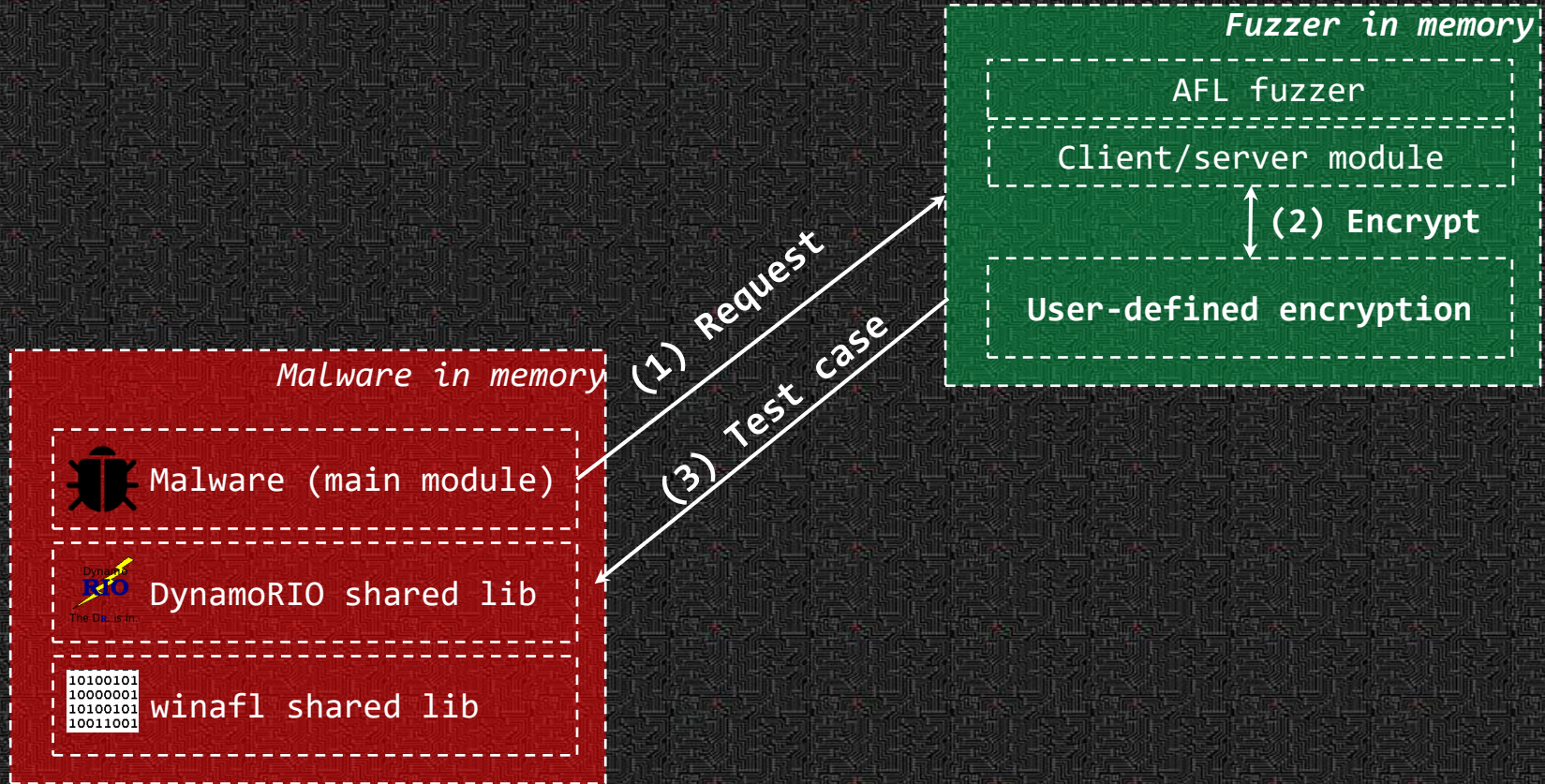
WinAFL patch



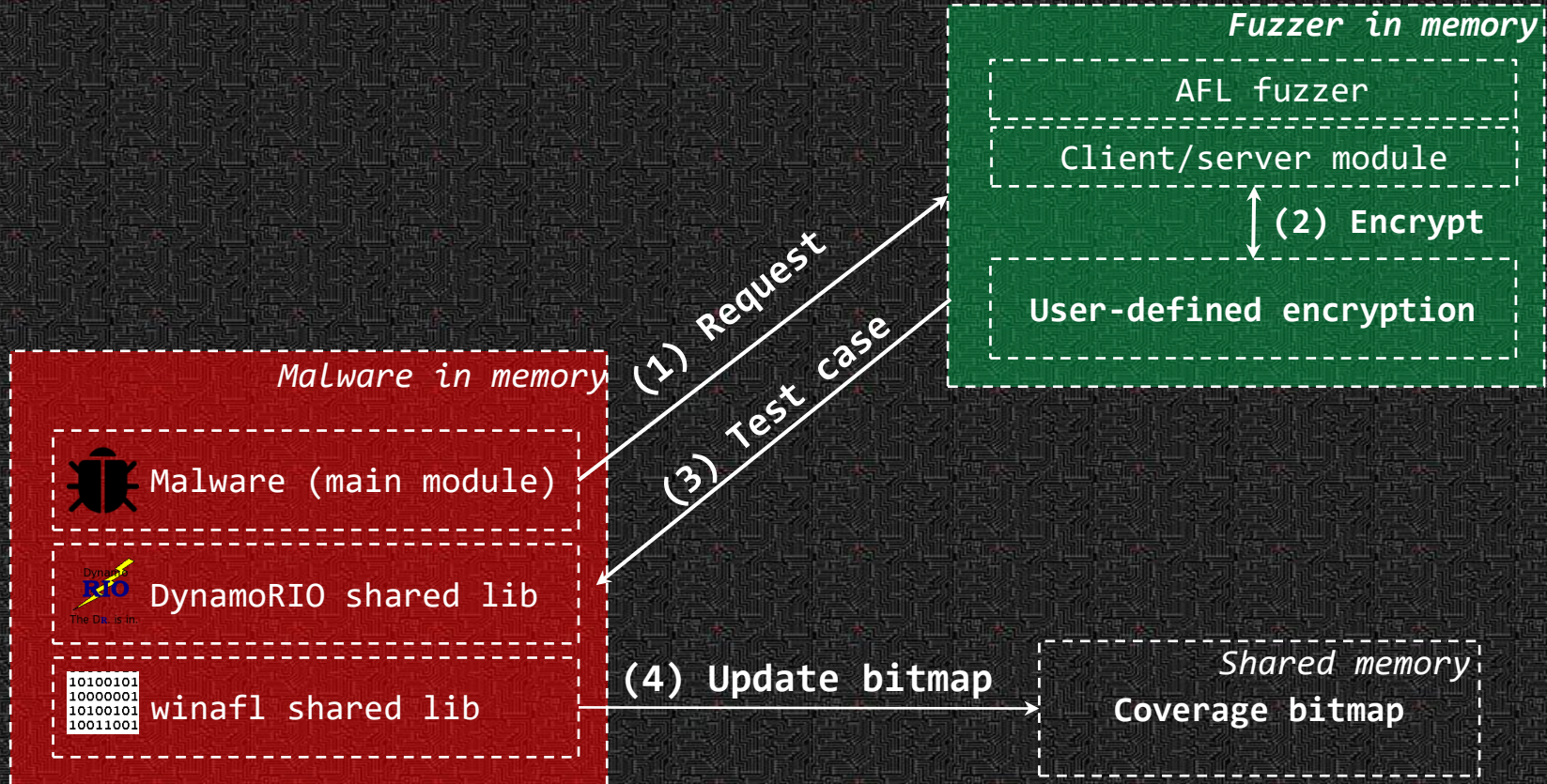
WinAFL patch



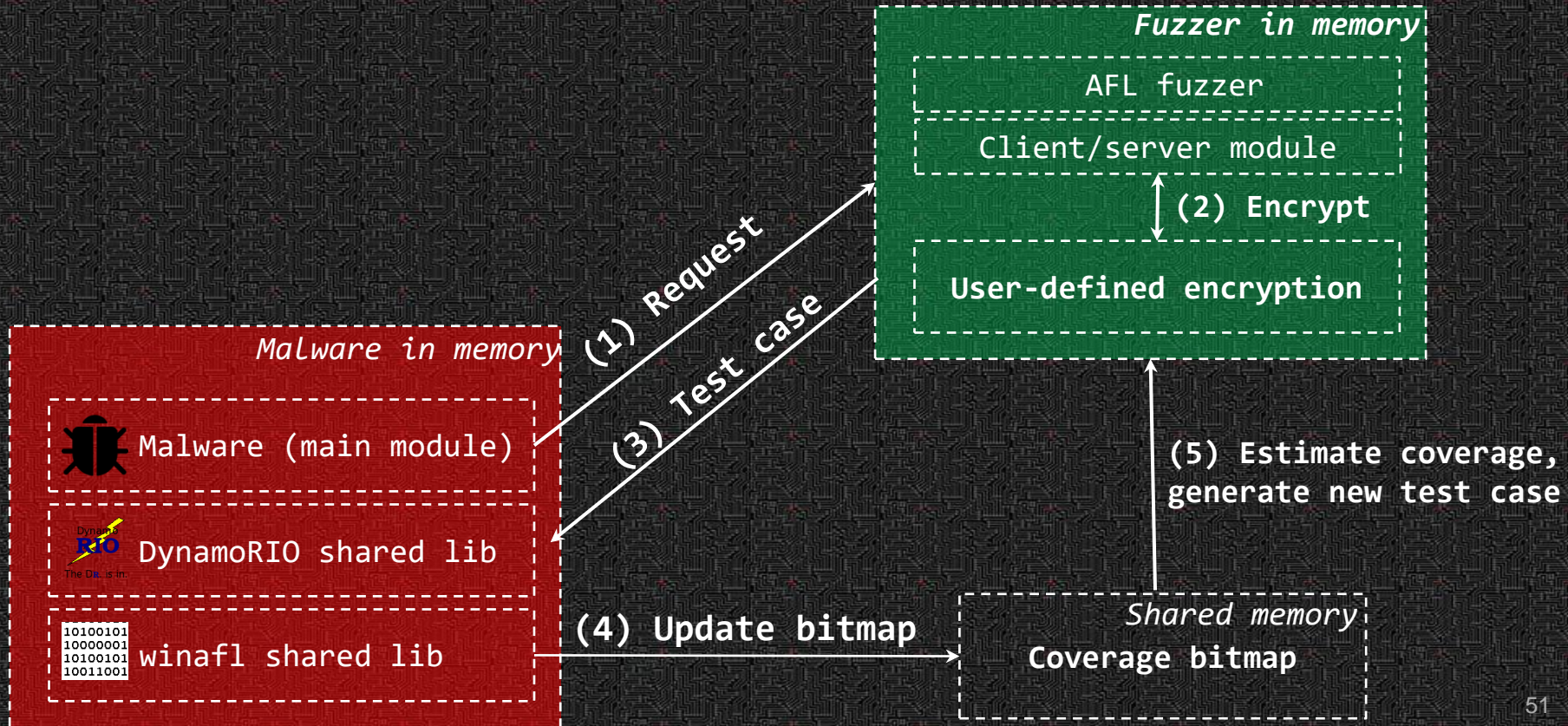
WinAFL patch



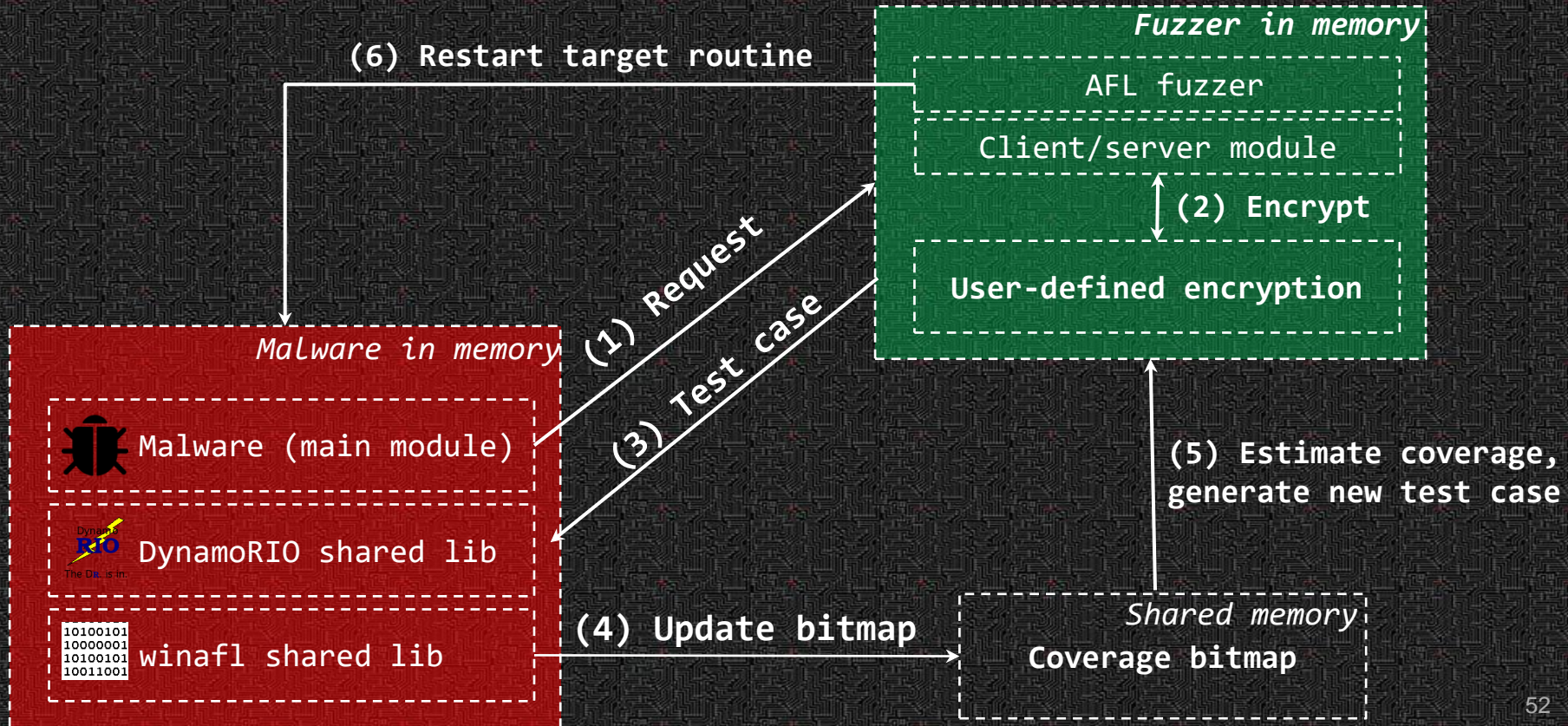
WinAFL patch



WinAFL patch



WinAFL patch



WinAFL patch. Usage

```
> afl-fuzz.exe -usage
```

User-defined custom test cases processing library:

-l - Path to library with user-defined CnC server

Fake server settings (custom_winafl_server.dll) via AFL_CUSTOM_DLL_ARGS

<port_id> - Server port to listen for incoming connections

Network fuzzing (custom_net_fuzzer.dll) provided via AFL_CUSTOM_DLL_ARGS

-a - IP address to send data in

-U - Use UDP (default TCP)

-p - Port to send data in

-w - Delay in milliseconds before sending data

WinAFL patch. User-defined C&C

- `dll_init()`
- `dll_run(char *data, long size, int fuzz_iterations)`
 - `data` - content of test case
 - `size` - size of test case
 - `fuzz_iterations` - defines a current fuzzing iteration number
- There is two examples distributed with winAFL

TOOL DEMO

4 hours after

```
C:\Windows\system32\cmd.exe

bit flips : 4/4528, 2/4507, 2/4465
byte flips : 0/566, 0/545, 0/504
arithmetics : 10/31.5k, 0/1530, 0/169
known ints : 4/3225, 3/18.3k, 1/20.1k
dictionary : 0/0, 0/0, 0/6322
havoc : 68/13.8k, 0/0
trim : 95.81%/219, 0.00%
levels : 6
pending : 71
pend fav : 2
own finds : 91
imported : n/a
stability : 87.34%
-----+-----
1 processes nudged-----+
SUCCESS: The process with PID 453968 has been terminated.

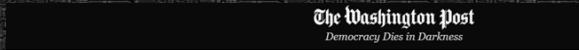
WinAFL 1.11 based on AFL 2.43b (posgrahber.exe)

+- process timing -----+- overall results -----+
| run time : 0 days, 4 hrs, 16 min, 4 sec | cycles done : 2 |
| last new path : 0 days, 0 hrs, 58 min, 13 sec | total paths : 92 |
| last uniq crash : 0 days, 2 hrs, 12 min, 17 sec | uniq crashes : 3 |
| last uniq hang : 0 days, 0 hrs, 56 min, 3 sec | uniq hangs : 20 |
+- cycle progress -----+- map coverage -----+
| now processing : 51 (55.43%) | map density : 0.37% / 0.48% |
| paths timed out : 0 (0.00%) | count coverage : 2.34 bits/tuple |
+- stage progress -----+- findings in depth -----+
| now trying : trim 256\256 | favored paths : 8 (8.70%) |
| stage execs : 17/50 (34.00%) | new edges on : 10 (10.87%) |
| total execs : 111k | total crashes : 8 (3 unique) |
| exec speed : 0.03/sec (zzzz...) | total tmouts : 335 (20 unique) |
+- fuzzing strategy yields -----+- path geometry -----+
| bit flips : 4/4528, 2/4507, 2/4465 | levels : 6 |
| byte flips : 0/566, 0/545, 0/504 | pending : 71 |
| arithmetics : 10/31.5k, 0/1530, 0/169 | pend fav : 2 |
| known ints : 4/3225, 3/18.3k, 1/20.1k | own finds : 91 |
| dictionary : 0/0, 0/0, 0/6322 | imported : n/a |
| havoc : 68/13.8k, 0/0 | stability : 87.34% |
```


Case Study I. Mirai

未来

Mirai. Overview



The Switch

Cyberattack that disrupted websites is under investigation

By Andrea Peterson

Make a contribution

Subscribe

Find a job

Sign in

Search

News

Opinion

Sport

Culture

Lifestyle

More

US World Environment Soccer US politics Business Tech Science Homelessness

The Guardian

US edition



Hacking

DDoS attack that disrupted internet was largest of its kind in history, experts say

Advertisement



U.S. OCTOBER 21, 2016 / 6:20 AM / 2 YEARS AGO

Cyber attacks disrupt PayPal, Twitter, other sites

Mirai. Overview

- IoT-based botnet DDoS
- Most disruptive DDoS cyber-attack in history
 - 2016 Dyn DDoS (1.2Tb/s).
 - Krebs on Security (620 Gb/s)
 - OVH DDoS (1TB/s)
- Hundreds of thousands devices across 164 countries
- Some elements of SDLC:

```
gcc -lefence -g -DDEBUG -static -lpthread -pthread -O3 src/*.c -o loader.dbg
```

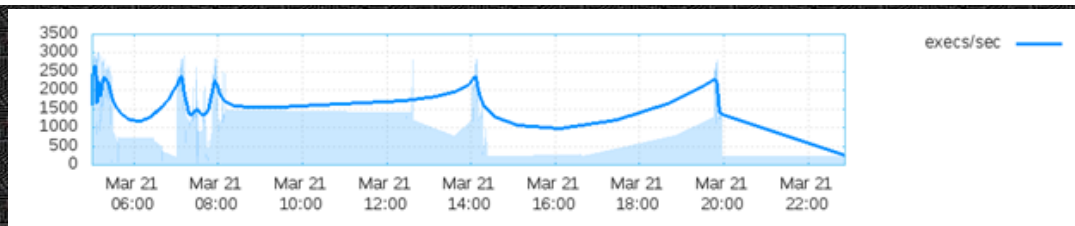
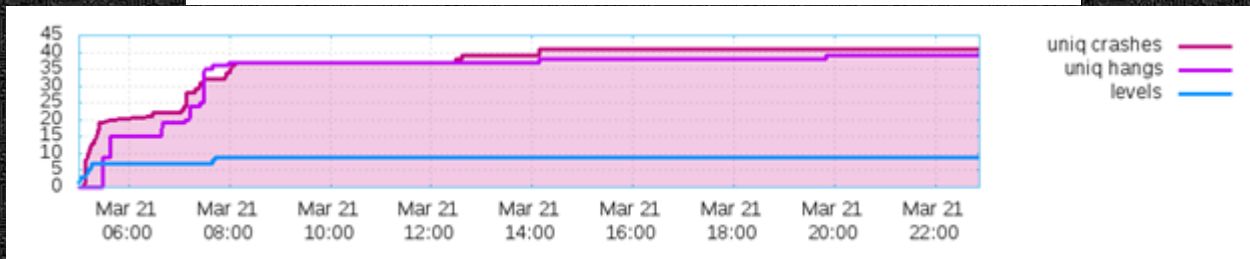
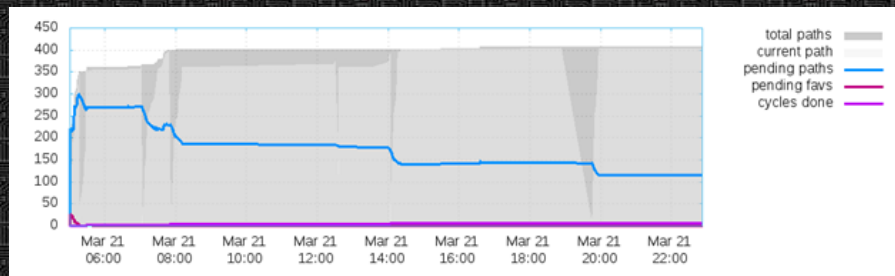

Mirai. HTTP-response parser

```
375     if (FD_ISSET(conn->fd, &fdset_rd))
376     {
377         if (conn->state == HTTP_CONN_RECV_HEADER)
378         {
379             int processed = 0;
380
381             util_zero(generic_memes, 10240);
382             if ((ret = recv(conn->fd, generic_memes, 10240, MSG_NOSIGNAL | MSG_PEEK)) < 1)
383             {
384                 close(conn->fd);
385                 conn->fd = -1;
386                 conn->state = HTTP_CONN_INIT;
387                 continue;
388             }
389
390
391             // we want to process a full http header (^:
392             if (util_memsearch(generic_memes, ret, "\r\n\r\n", 4) == -1 && ret < 10240)
393                 continue;
```


Mirai. Seed File

```
GET / HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/51.0.2704.103 Safari/537.36
Host: localhost
Connection: keep-alive
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.8
Content-Type: application/x-www-form-urlencoded
content-length: 3
```


Mirai. Fuzzing Statistics



Mirai. Vulnerability

```
else if (loc_ptr[0] == '/')
{
    //handle relative url
    util_zero(conn->path + 1, HTTP_PATH_MAX - 1);
    if (util_strlen(&(amp;loc_ptr[ii + 1])) > 0 && util_strlen(&(amp;loc_ptr[ii + 1])) < HTTP_PATH_MAX)
        util_strecpy(conn->path + 1, &(loc_ptr[ii + 1]));
}

conn->state = HTTP_CONN_RESTART;
continue;
}
```


Mirai. Crash Case

GET / HTTP/1.1

User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36

Host: **location:/keep-alive**

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/*/*;q=0.8

Accept-Language: =0.8

Content-Type: application/x-www-form-urlencoded

content-length: 3

LOL

Mirai. Exploitation Demo

Case Study II. Dexter v2

Dexter. Overview

- Point-of-sales (PoS) malware which is targeted Microsoft Windows terminals
- Steals credit/debit card details
- First known botnet that targets POS terminals (mostly in US)



Dexter.Target

```
if(HttpSendRequest(hRequest,"Content-Type:application/x-www-form-urlencoded")

//Build cookie url
memset(Url,0x00,sizeof(Url));
wsprintf(Url,"http://%s%s",Urls[x],Pages[x]);
////////////////////

//Get cookie - commands
memset(Commands,0x00,sizeof(Commands));
if(GetCookie(Url,Commands)==TRUE) { //We are on valid url

    pCommands = Commands;
    pCommands += strlen(response);
    //MessageBox(NULL,pCommands,"Check if valid",MB_OK);
    if(*pCommands=='$') { //This seems to be real command

        ExecCommands(pCommands);
        ValidIndex = TRUE;
        AlreadyConnected = TRUE;
        PersistInfo = FALSE;
    } else { ValidIndex = FALSE; }
} else { ValidIndex = FALSE; }
}
```

```
void ExecCommands(char *pCommands) {

    char Url[255],val[5];
    DWORD dVal;

    pCommands++; //skip '$'
    //MessageBox(NULL,pCommands,NULL,MB_OK);
    while(*pCommands!='#' && strlen(pCommands)) {

        if(StrCmpNI(pCommands,update,strlen(update))==0) {

            pCommands += strlen(update);
            CopyTill(Url,pCommands,',');
            strcat(Url,varKey);
            strcat(Url,Key);
            Update(Url);
        } else
        if(StrCmpNI(pCommands,checkin,strlen(checkin))==0) {

            pCommands += strlen(checkin);
            pCommands += CopyTill(val,pCommands,',');
            pCommands += 1;
        }
    }
}
```

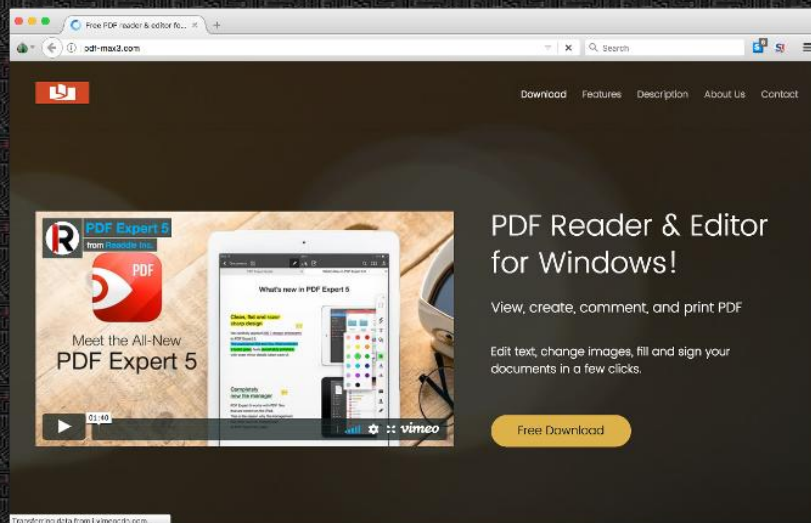

Dexter. Vulnerability

```
void ExecCommands(char *pCommands) {  
  
    char Url[255],val[5];  
    DWORD dVal;  
  
    pCommands++; //skip '$'  
    ///MessageBox(NULL,pCommands,NULL,MB_OK);  
    while(*pCommands!='#' && strlen(pCommands)) {  
  
        if(StrCmpNI(pCommands,update,strlen(update))==0) {  
  
            pCommands += strlen(update);  
            CopyTill(Url,pCommands,',');  
            strcat(Url,varKey);  
            strcat(Url,Key);  
            Update(Url);  
        } else  
        if(StrCmpNI(pCommands,checkin,strlen(checkin))==0) {  
  
            pCommands += strlen(checkin);  
            pCommands += CopyTill(val,pCommands,',');  
            pCommands += 1;  
        }  
    }  
}
```


Case Study III. TinyNuke

TinyNuke. Overview

- Man-in-the-browser Trojan equipped with common features: WebInjects, SOCKS, Proxy, JSON parsers and etc.
- Distributed over trojanized PDF Reader



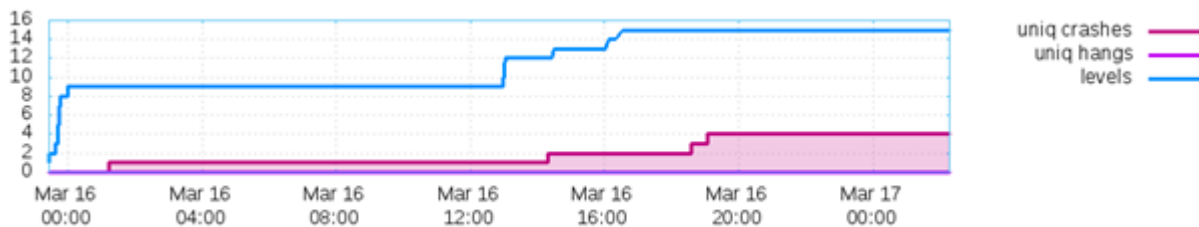
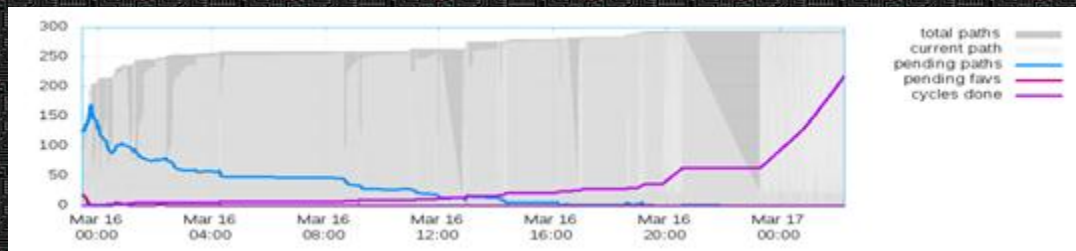
TinyNuke. Target

```
void LoadWebInjects()
{
    if (loaded)
        return;
    char request[32] = { 0 };
    Funcs::pLstrcpyA(request, Strs::injectsRequest);
    char *jsonStr = PanelRequest(request, NULL);
    if (!(json = AiJsonParse(jsonStr)))
        goto err;
    if (json->error != AI_JSON_E_OK)
        goto err;
    if (json->root.type != AI_JSON_OBJECT)
        goto err;
    loaded = TRUE;
    return;
err:
    Funcs::pFree(jsonStr);
    AiJsonDestroy(json);
    Funcs::pSleep(POLL);
    LoadWebInjects();
}
```


TinyNuke. Seed File

```
{
  "expand" : "attributes",
  "link" : {
    "rel" : "self",
    "href" : "http://localhost:8095/crowd/rest/usermanagement/1/user?username=my_username"
  },
  "name" : "my_username",
  "first-name" : "My",
  "last-name" : "Username",
  "display-name" : "My Username",
  "email" : "user@example.test",
  "password" : {
    "link" : {
      "rel" : "edit",
      "href" : "http://localhost:8095/crowd/rest/usermanagement/1/user/password?username=my_username"
    }
  },
  "active" : true,
  "attributes" : {
    "link" : {
      "rel" : "self",
      "href" : "http://localhost:8095/crowd/rest/usermanagement/1/user/attribute?username=my_username"
    },
    "attributes" : []
  }
}
```


TinyNuke. Statistics



TinyNuke. Vulnerability

```
364 AiJsonError ParseValue(AiJson *json, char **str, AiJsonValue *value)
365 {
366     AiJsonError err;
367     switch(**str)
368     {
369         case CHAR_STR_OPEN_CLOSE:
370         {
379         case CHAR_OBJECT_OPEN:
380         {
390         case CHAR_ARRAY_OPEN:
391         {
392             value->type = AI_JSON_ARRAY;
393             value->data.array = JsonListCreate(json);
394             if(!value->data.array)
395                 return AI_JSON_E_ALLOC;
396             err = ParseArray(json, str, value->data.array);
397             if(err != AI_JSON_E_OK)
398                 return err;
399             break;
400         }
```


TinyNuke. Crash Case

[illegible]

DrItrace

- DrItrace is an open-source API calls tracer for Windows (similar to ltrace for Linux).

```
drItrace.exe -logdir . -print_ret_addr - malware.exe
```

```
234369  ~~2840~~ WINHTTP.dll!WinHttpConnect
234370      arg 0: 0x003ca440 (type=<unknown>, size=0x0)
234371      arg 1: susiku.info (type=wchar_t*, size=0x0)
234372      arg 2: 0x00000050 (type=<unknown>, size=0x0)
234373      arg 3: 0x0 (type=DWORD, size=0x4)
234553  ~~2840~~ WINHTTP.dll!WinHttpOpenRequest
234554      arg 0: 0x004173a0 (type=<unknown>, size=0x0)
234555      arg 1: GET (type=wchar_t*, size=0x0)
234556      arg 2: /rbody320 (type=wchar_t*, size=0x0)
234557      arg 3: <null> (type=wchar_t*, size=0x0)
234558      arg 4: <null> (type=wchar_t*, size=0x0)
234559      arg 5: <null> (type=wchar_t*, size=0x0)
```


Conclusion

- Bugs in malware exist and can be used to defend against them
- Coverage-guided fuzzing was able to find bugs in each malware selected for experiment within 24 hours
- Two bugs lead to RCE, one bug can be used to defend against DDoS
- This technique can also be used to find bugs in network-based applications (probably most useful application)

Thank you!

[@MShudrak](#)

<https://github.com/mxmssh>

<https://www.linkedin.com/in/mshudrak>