Netlab 360.com

Security Visibility

Tracking Mirai Variants

Ya Liu (*speaker*)

Hui Wang

Outline

- Background
- Data and methodology
  - Configuration
  - Supported attack methods
- Analysis of typical Mirai branches
  - MASUTA, OWARI, WICKED
- (*newly added*) Classifications on *VessOnSecurity's* samples

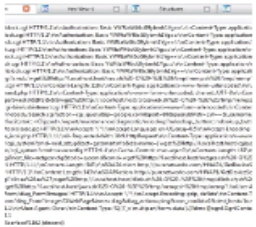
A short history of Mirai

- Firstly blogged by *@MalwareMustDie* in August, 2016
- Getting known for crippling Krebsonsecurity, OVH, and DYN in autumn 2016
- Source was released on Sep 30, 2016
- Dozens of variants got derived, some of them were also open sourced

Mirai variants

360 Netlab
@360Netlab

We started to see this about 14 hours ago as well, actually we have counted 16+1 exploits being used, pretty crazy.



Rommel @rommeljoven17
Owari, a Mirai variant, added 2 exploits from 2017
-EnGenius RCE
-NetGain Ping Command Injection
At least 14 exploits are now used

显示这个主题帖

翻译推文

下午11:02 - 2018年9月7日

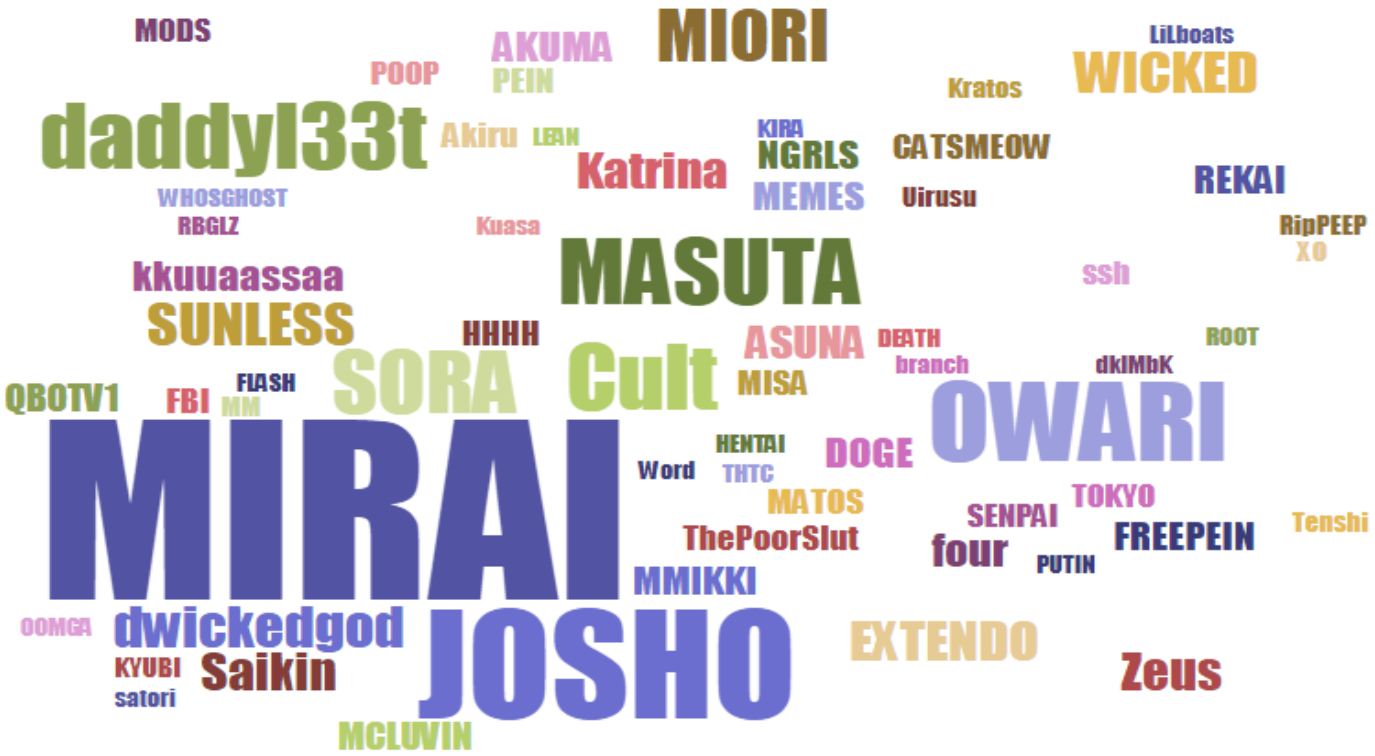
Mirai variants and branches

- Mirai variants are usually named and classified according to their branch names

- Branch refers to the command used in infection
  - "*/bin/busybox MIRAI*"
  - "*MIRAI: applet not found*"

- Branches were often replaced with new words in later variants
  - E.g., AKUMA, OWARI, MASUTA, SATORI

Branch word cloud



Fine-grained classification is needed

- It's common that the same branch of samples vary a lot
  - **7** attack methods and **32** configurations in sample of *63e7878d0a9877fdcea6e094cb291ed5*
  - **10** attack methods and **196** configurations in sample of *a67c1814f5f558b10d11c312b2e2113a*

- For better variant tracking, more fine-grained classifications are needed

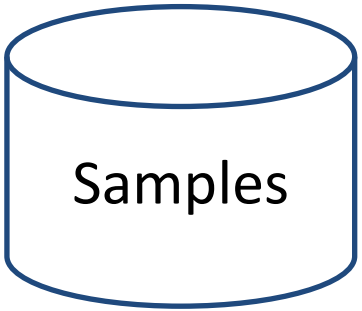
Outline

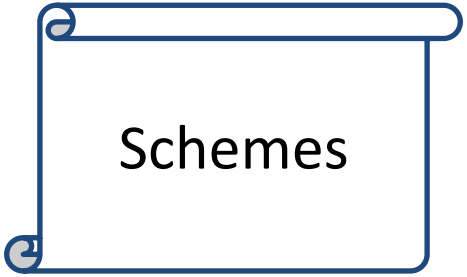
- Background

- Data and methodology
  - Configuration
  - Supported attack methods

- Analysis of typical Mirai branches
  - MASUTA, OWARI, WICKED

Our solution





- **10,784** samples of x86 & ARM

- Configurations
- Attack methods
- Usernames and passwords

- **4** classification schemes

Data extraction model

- Static analysis
  - To find target functions in sample

IDAPython

- Dynamic analysis
  - To emulate the found functions to obtain interested data

Unicorn

The ultimate CPU emulator

- Synthesis

A configuration example

```
[0x01]: "\x059", size=2
[0x02]: "\x07\xbe", size=2
[0x03]: "DaddyL33T Infected Your Shit\x00", size=29
[0x04]: "shell\x00", size=6
[0x05]: "enable\x00", size=7
[0x06]: "system\x00", size=7
[0x07]: "sh\x00", size=3
[0x08]: "/bin/busybox JOSHO\x00", si
[0x09]: "JOSHO: applet not found\x00
[0x0a]: "ncorrect\x00", size=9
[0x0b]: "/bin/busybox ps\x00", size=
[0x0c]: "/bin/busybox kill -9 \x00", size=22
[0x0d]: "/proc/\x00", size=7
```

CNC & report ports

prompt line

Scanner parameters

Infection verification

Killer parameters

.....

MD5=0ae272306d313c6abf1433b85e0a2352

A summary of Mirai configuration

- A self-defined database to store running parameters
  - Composed of variable number of items
  - Each item is uniquely indexed
  - Items are stored encrypted in an XOR-encryption
- Varying greatly among variants in terms of encryption key, item count, and content
- A good entry point for variant classification

About configuration extraction

- 2 functions to analyze: **table_init()** and **resolve_cnc_addr()**
- 4 scripts: 2 for static analysis and 2 for dynamic analysis
  - **table_init**() : an array of {slot_addr, cipher-text, size}
  - **resolve_cnc_addr**(): indexes of CNC server and port
- Key is brute-force searched in the space of 1~256
- The final result is an array of {index, plain-text, size}

About table_init()

```
void table_init(void)
{
    add_entry(TABLE_CNC_DOMAIN, "\x41\x4C\x41\x0C\x41\x4A\x43\x4C\
    add_entry(TABLE_CNC_PORT, "\x22\x35", 2);    // 23

    add_entry(TABLE_SCAN_CB_DOMAIN, "\x50\x47\x52\x4D\x50\x56\x0C\
    add_entry(TABLE_SCAN_CB_PORT, "\x99\xC7", 2);          // 48101
```

index cipher-text item-size

```
static void add_entry(uint8_t id, char *buf, int buf_len)
{
    char *cpy = malloc(buf_len);

    util_memcpy(cpy, buf, buf_len);

    table[id].val = cpy;
    table[id].val_len = (uint16_t)buf_len;
#ifdef DEBUG
    table[id].locked = TRUE;
#endif
}
```

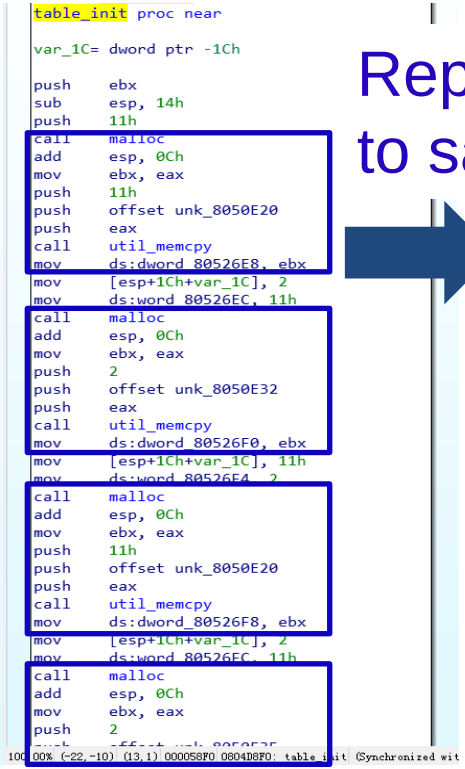
cipher text is copied to a new memory block

config DB

slot is determined by the index

Binary table_init()

A big function with a single and big instruction block



Repeatedly calling malloc/util_memcpy to save individual configuration items

```
push    11h
call    malloc
add     esp, 0Ch
mov     ebx, eax
push    11h
push    offset unk_8050E20
push    eax
call    util_memcpy
mov     ds:dword_80526E8, ebx
```

item size

cipher text address

slot address

About resolve_cnc_addr ()

called in main() or anti_gdb_entry()

```
static void resolve_cnc_addr(void)
{
    struct resolv_entries *entries;
```

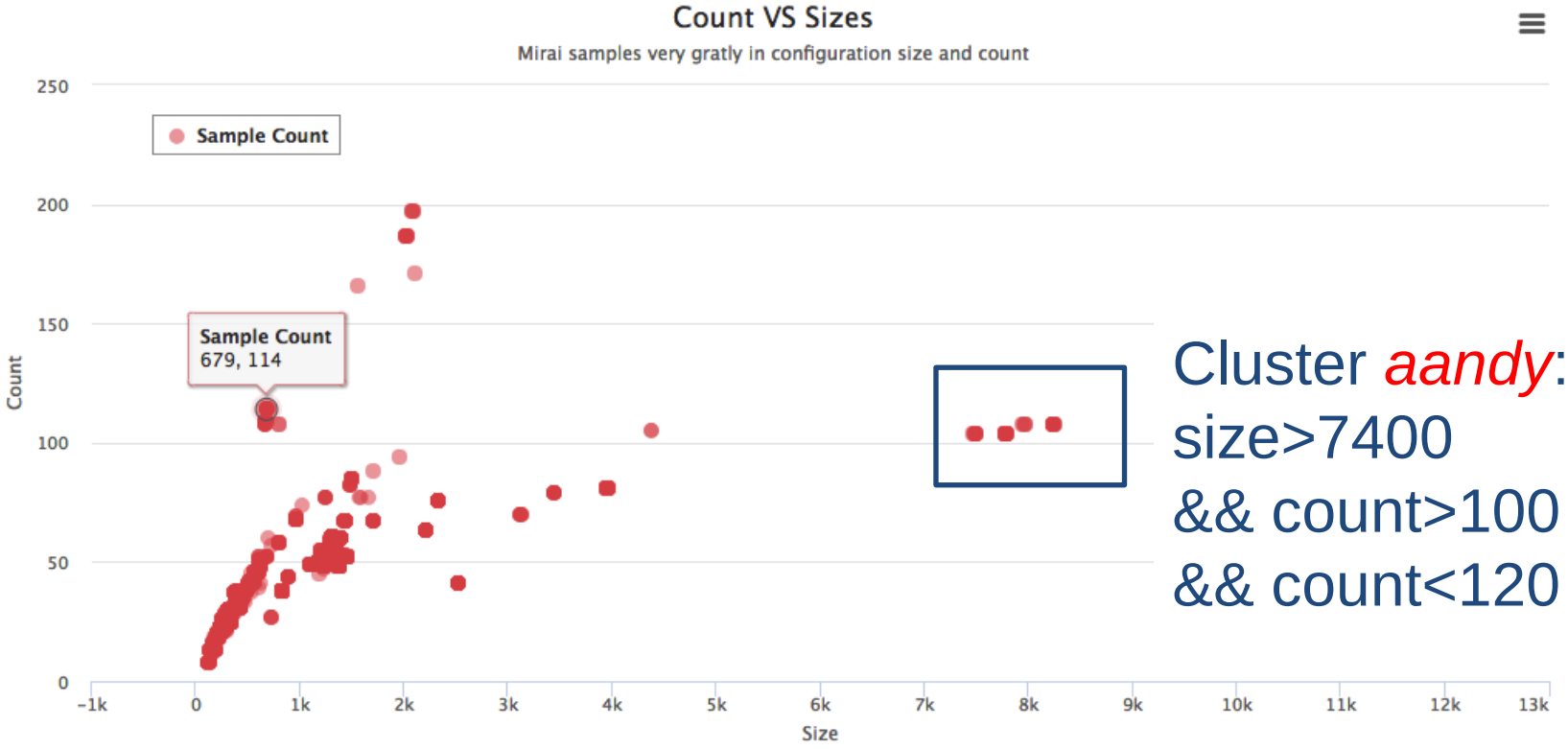
retrieve & resolve CNC domain/IP

```
table_unlock_val(TABLE_CNC_DOMAIN);
entries = resolv_lookup(table_retrieve_val(TABLE_CNC_DOMAIN, NULL))
table_lock_val(TABLE_CNC_DOMAIN);
srv_addr.sin_addr.s_addr = entries->addrs[rand_next() % entries->ad
resolv_entries_free(entries);
```

```
table_unlock_val(TABLE_CNC_PORT);
srv_addr.sin_port = *((port_t *)table_retrieve_val(TABLE_CNC_PORT,
table_lock_val(TABLE_CNC_PORT);
```

retrieve CNC port

Scheme-1: clustering based on config count/size



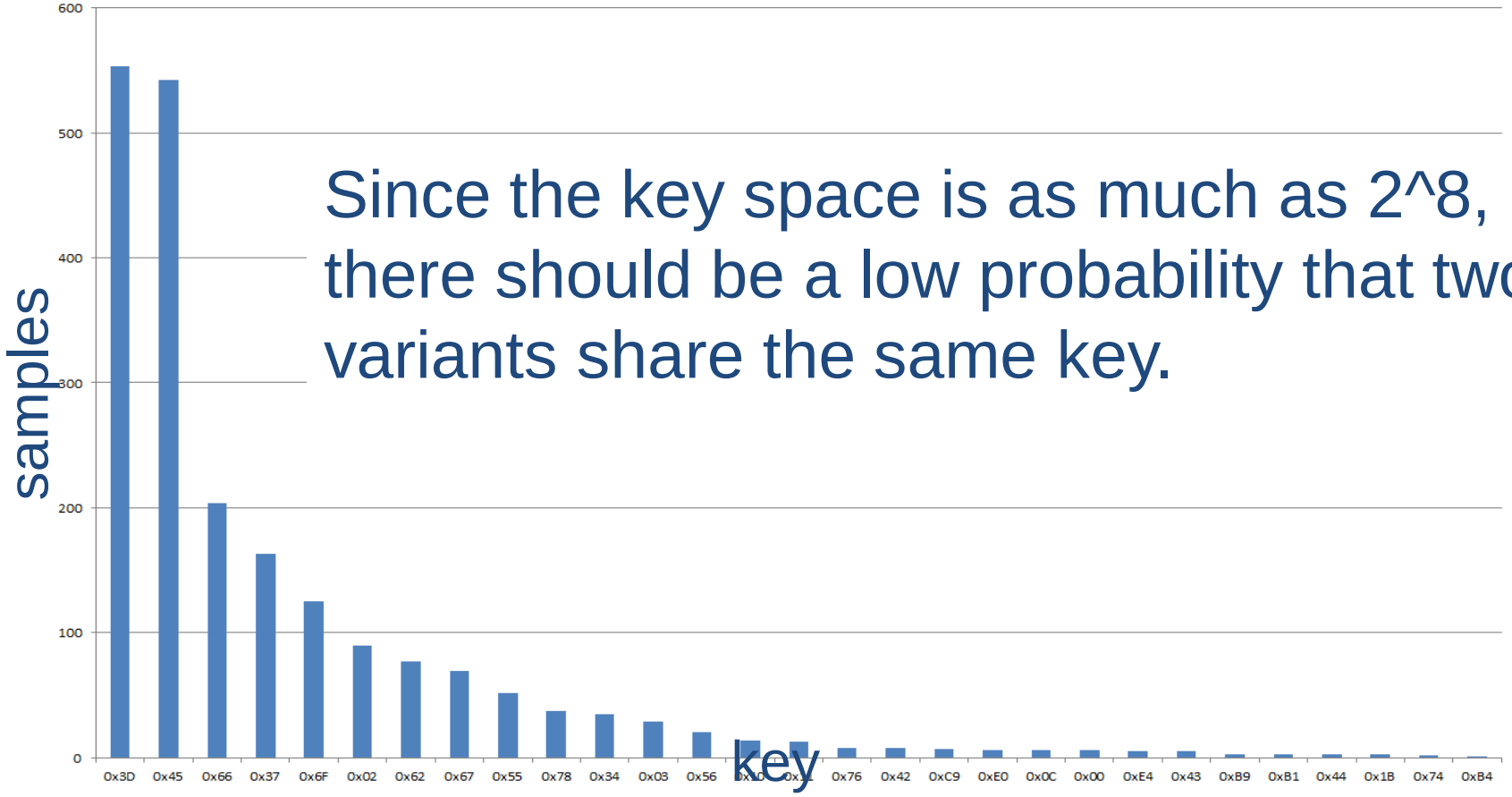
Cluster *aandy*:
size>7400
&& count>100
&& count<120

Cluster *aandy*

Branch name	Key	C2	Samples
KYUBI	0x34	cnc.aandy.xyz	4
MIRAI	0x34	cnc.aandy.xyz	8
MIRAI	0x34	www.aandy.cf	7
MIRAI	0x34	www.askjasghasg.ru	16

107.179.126.64

Scheme-2: key based classification



Since the key space is as much as 2^8, there should be a low probability that two variants share the same key.

A use case

- The branches of **OWARI**, **JOSHO**, and **Cult** can be connected via the key of **0x54**

MD5	Branch	Configuration count/size	CNC
0729b89281c831fc035d56fbf14631da	Cult	30/333	198.134.120.150
23a98fc659982da993e7825eb87bb640	OWARI	30/340	198.134.120.150
2ff2d4feff4ffcec355f52993ce7b73e	JOSHO	30/346	198.134.120.150

Supported attack methods

- It's reasonable to classify DDoS botnet variants based on their supported attack methods

- Mirai variants did vary a lot in attack methods
  - 10 attack methods were found in the released code
  - Dozens of new, or updated, methods have been detected in later variants

- The classification data includes method count, type, and command code

Attack method initialization

```
BOOL attack_init(void)
{
    int i;

    add_attack(ATK_VEC_UDP,       (ATTACK_FUNC)attack_udp_generic);
    add_attack(ATK_VEC_VSE,       (ATTACK_FUNC)attack_udp_vse);
    add_attack(ATK_VEC_DNS,       (ATTACK_FUNC)attack_udp_dns);
    add_attack(ATK_VEC_UDP_PLAIN, (ATTACK_FUNC)attack_udp_plain);

    add_attack(ATK_VEC_SYN,       (ATTACK_FUNC)attack_tcp_syn);
    add_attack(ATK_VEC_ACK,       (ATTACK_FUNC)attack_tcp_ack);
    add_attack(ATK_VEC_STOMP,     (ATTACK_FUNC)attack_tcp_stomp);

    add_attack(ATK_VEC_GREIP,     (ATTACK_FUNC)attack_gre_ip);
    add_attack(ATK_VEC_GREETH,    (ATTACK_FUNC)attack_gre_eth);

    //add_attack(ATK_VEC_PROXY, (ATTACK_FUNC)attack_app_proxy);
    add_attack(ATK_VEC_HTTP,      (ATTACK_FUNC)attack_app_http);
```

command code

attack function

Inside of add_attack()

```
static void add_attack(ATTACK_VECTOR vector, ATTACK_FUNC func)
{
    struct attack_method *method = calloc(1, sizeof (struct attack_method));

    method->vector = vector;
    method->func = func;

    methods = realloc(methods, (methods_len + 1) * sizeof (struct attack_method *));
    methods[methods_len++] = method;
}
```

each method is allocated separate item

item is saved to method table

method table

Attack method extraction

- **attack_init()** is found according to characteristics of :
  - Composed of single instruction block
  - 1~2 unique functions are repeatedly called
  - Multiple (attack) functions are used as callbacks

- Command codes and attack functions are obtained by dynamic emulation

- The final result is an array of {cmd_code, atk_func}

Scheme-3: command code based classification

Command code combination	Samples
0_1_2_3_4_5_6_7_9_10	4488
0_1_2_3_4_5_6_7_8_9_10	3890
0_1_2_3_4_5_6_7_8	976
0_1_2_3_4_5_6_7_8_9	353
0_1_2_3_6_7_8	138
0_1_2_3_4_5_6_7_9	96
0_1_2_3_4	94
0_1_2_3	75
0_1_2_3_4_5_6_7_9_10_11_12	51
0_1_2	48

Fingerprinting attack functions

- To figure out extracted attack functions' real semantics
  - E.g., SYN-/UDP-/HTTP-flood

- It's inspired by the following 2 findings:
  - A set of attack options, together with command codes, were defined to deliver attack parameters
  - Different attack functions usually use different options

Mirai attack options

```
#define ATK_OPT_PAYLOAD_SIZE     0    // What should the size of the packet data be?
#define ATK_OPT_PAYLOAD_RAND     1    // Should we randomize the packet data contents?
#define ATK_OPT_IP_TOS           2    // tos field in IP header
#define ATK_OPT_IP_IDENT         3    // ident field in IP header
#define ATK_OPT_IP_TTL           4    // ttl field in IP header
#define ATK_OPT_IP_DF            5    // Dont-Fragment bit set
#define ATK_OPT_SPORT            6    // Should we force a source port? (0 = random)
#define ATK_OPT_DPORT            7    // Should we force a dest port? (0 = random)
#define ATK_OPT_DOMAIN           8    // Domain name for DNS attack
#define ATK_OPT_DNS_HDR_ID       9    // Domain name header ID
//#define ATK_OPT_TCPCC          10 // TCP congestion control
#define ATK_OPT_URG              11   // TCP URG header flag
#define ATK_OPT_ACK              12   // TCP ACK header flag
#define ATK_OPT_PSH              13   // TCP PSH header flag
#define ATK_OPT_RST              14   // TCP RST header flag
#define ATK_OPT_SYN              15   // TCP SYN header flag
#define ATK_OPT_FIN              16   // TCP FIN header flag
#define ATK_OPT_SEQRND           17   // Should we force the sequence number? (TCP only)
#define ATK_OPT_ACKRND           18   // Should we force the ack number? (TCP only)
#define ATK_OPT_GRE_CONSTIP      19   // Should the encapsulated destination address be the same as the target?
#define ATK_OPT_METHOD           20   // Method for HTTP flood
#define ATK_OPT_POST_DATA        21   // Any data to be posted with HTTP flood
#define ATK_OPT_PATH             22   // The path for the HTTP flood
#define ATK_OPT_HTTPS            23   // Is this URL SSL/HTTPS?
#define ATK_OPT_CONNS            24   // Number of sockets to use
#define ATK_OPT_SOURCE           25   // Source IP
```

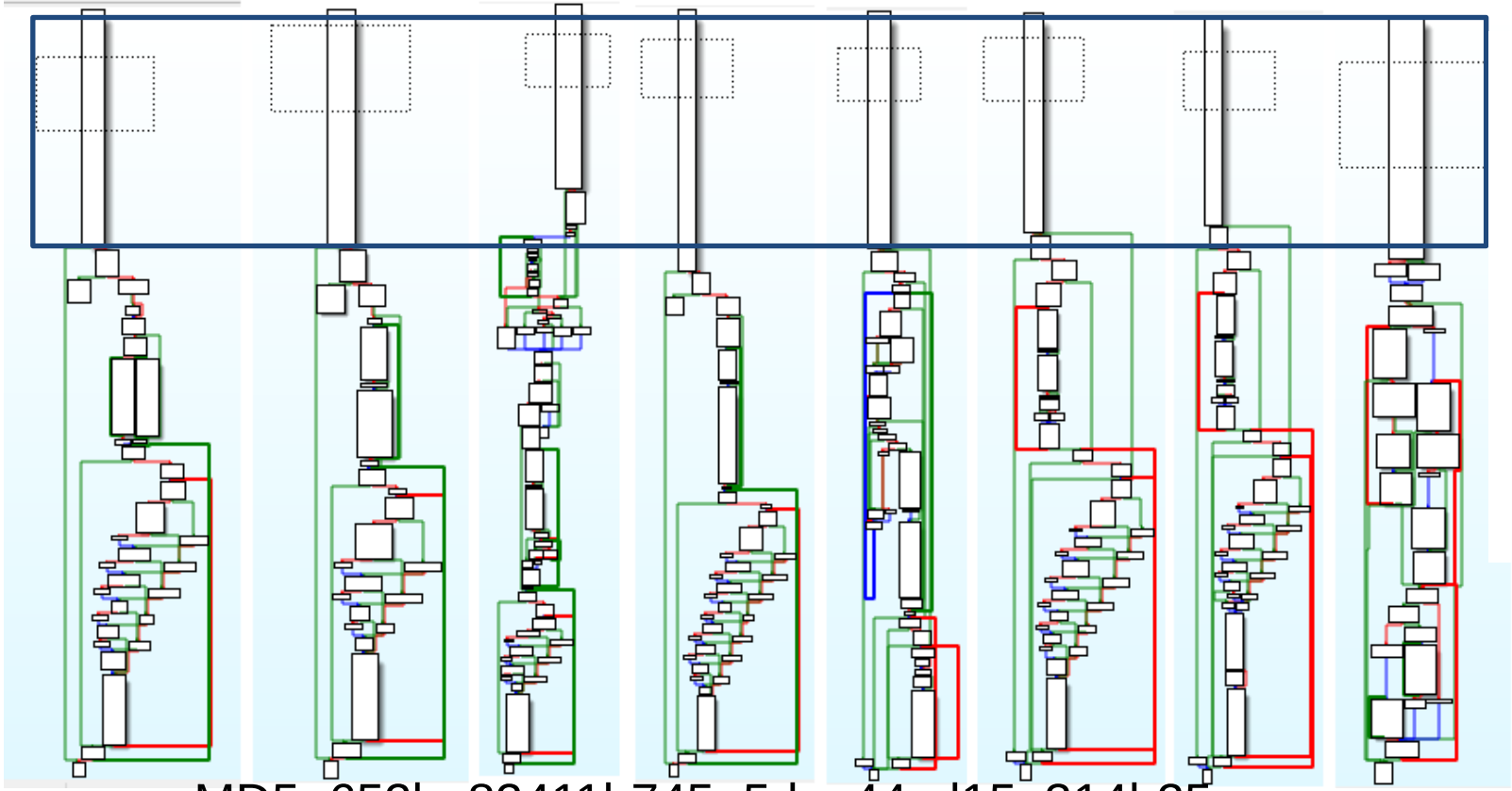
Option set is specific to attack type

```
void attack_app_http(uint8_t targs_len, struct attack_target *targs, uint8_t opts_len, struct attack_option *opts)
{
    int i, ii, rfd, ret = 0;
    struct attack_http_state *http_table = NULL;
    char *postdata = attack_get_opt_str(opts_len, opts, ATK_OPT_POST_DATA, NULL);
    char *method = attack_get_opt_str(opts_len, opts, ATK_OPT_METHOD, "GET");
    char *domain = attack_get_opt_str(opts_len, opts, ATK_OPT_DOMAIN, NULL);
    char *path = attack_get_opt_str(opts_len, opts, ATK_OPT_PATH, "/");
    int sockets = attack_get_opt_int(opts_len, opts, ATK_OPT_CONNS, 1);
    port_t dport = attack_get_opt_int(opts_len, opts, ATK_OPT_DPORT, 80);
```

different functions,
different option sets

```
void attack_gre_ip(uint8_t targs_len, struct attack_target *targs, uint8_t opts_len, struct attack_option *opts)
{
    int i, fd;
    char **pkts = calloc(targs_len, sizeof (char *));
    uint8_t ip_tos = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_TOS, 0);
    uint16_t ip_ident = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_IDENT, 0xffff);
    uint8_t ip_ttl = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_TTL, 64);
    BOOL dont_frag = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_DF, TRUE);
    port_t sport = attack_get_opt_int(opts_len, opts, ATK_OPT_SPORT, 0xffff);
    port_t dport = attack_get_opt_int(opts_len, opts, ATK_OPT_DPORT, 0xffff);
    int data_len = attack_get_opt_int(opts_len, opts, ATK_OPT_PAYLOAD_SIZE, 512);
    BOOL data_rand = attack_get_opt_int(opts_len, opts, ATK_OPT_PAYLOAD_RAND, TRUE);
    BOOL gcip = attack_get_opt_int(opts_len, opts, ATK_OPT_GRE_CONSTIP, FALSE);
    uint32_t source_ip = attack_get_opt_int(opts_len, opts, ATK_OPT_SOURCE, LOCAL_ADDR);
```

Binary attack method functions



MD5=652ba82411b745e5dac44cd15e314b25

All start With a big instruction block

Fingerprinting definition

- FP(atk_func)={concatenation of option codes}
  - E.g., FP(attack_app_http)=0x15_0x14_0x08_0x16_0x18_0x07
- In total **43** unique fingerprints have been found
  - Most of them are shared across variants
- Maps of {FP, atk_type} could be established by manual RE or using symbols from unstripped samples

Scheme-4: attack type based classification

- Variant is defined as the coded attack types
  - E.g., {0-atk_udp1, 1-atk_udp_vse1, 2-atk_tcp_syn1, …}
- Information of method count, command codes, and attack types is fully exploited
- In total **126** unique combinations have been found

Outline

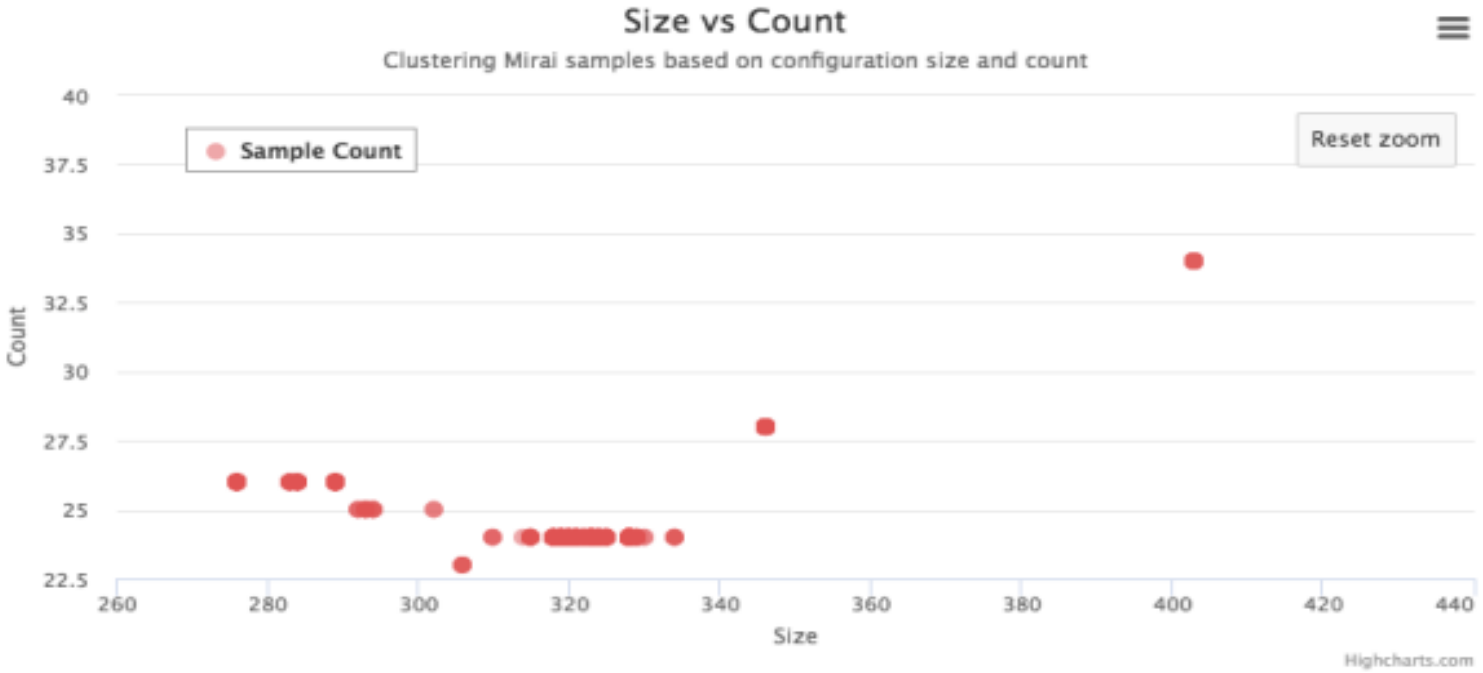
- Background
- Data and methodology
- Analysis of typical branches
  - MASUTA
  - OWARI
  - WICKED

MASUTA samples under scheme-2

- **4** keys have been found in MASUTA samples

Variant	Samples	CNCs
MASUTA+0x45	351	53
MASUTA+0x02	90	5
MASUTA+0x22	9	1
MASUTA+0x55	8	1

MASUTA+0x45 under scheme-1



MASUTA+0x45 under scheme-4

- Totally **8** combinations were found

samples	coded attack fingerprints
1	{0-atk_udp3, 1-atk_udp_vse1, 2-atk_udp_dns,
1	{0-atk_udp3, 1-atk_udp_vse1, 2-atk_udp_dns,
3	{0-atk_tcp_syn1, 1-atk_tcp_ack1, 2-atk_tcp_
8	{0-atk_udp1, 1-atk_udp_vse1, 2-atk_udp_dns,
11	{0-atk_udp_or_gre2, 1-atk_udp_vse1, 2-atk_u
20	{0-atk_udp_or_gre2, 1-atk_udp_vse1, 2-atk_u
64	{0-atk_udp_or_gre2, 1-atk_udp_vse1, 2-atk_u
236	{0-atk_udp_or_gre2, 1-atk_udp_vse1, 2-atk_u

Outline

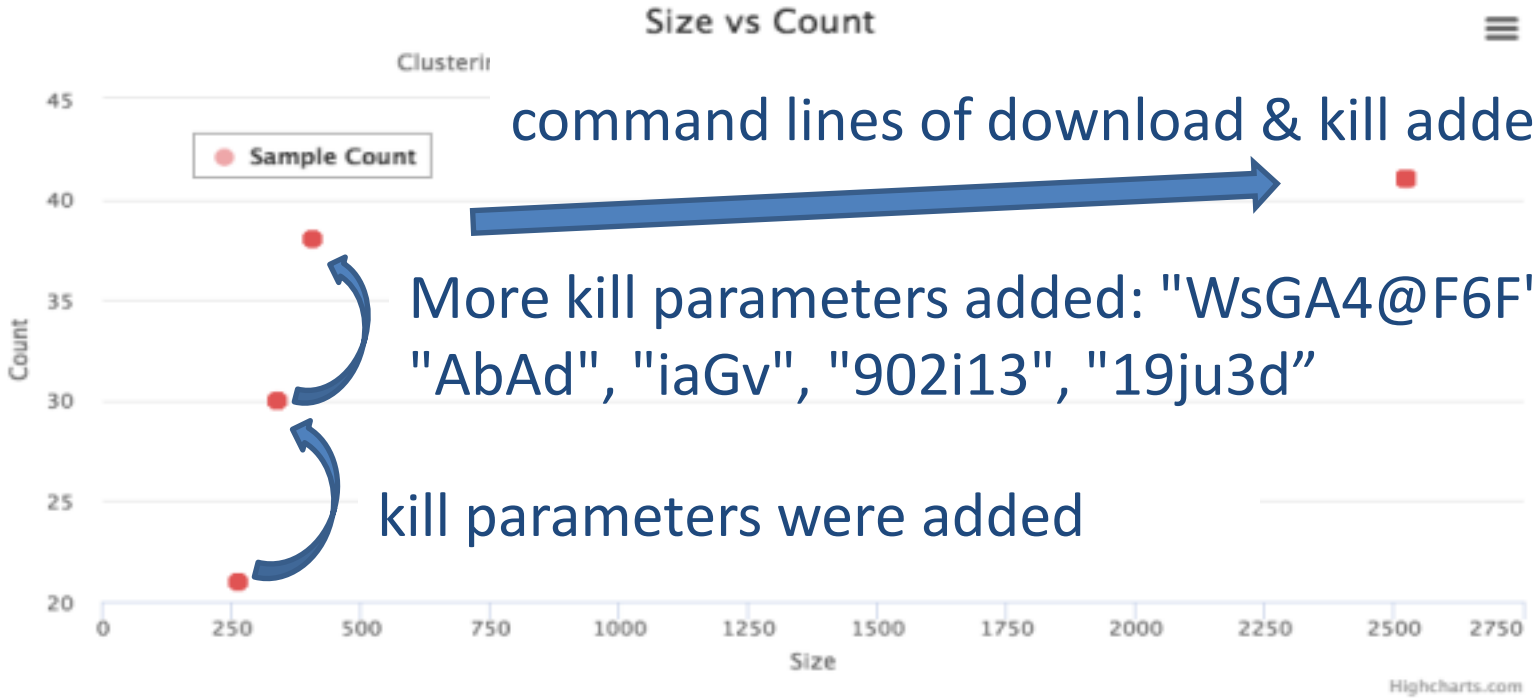
- Background
- Data and methodology
- Analysis of typical branches
  - MASUTA
  - OWARI
  - WICKED

OWARI samples under scheme-2

- **2** keys are found in OWARI samples

Variant	Samples	CNCs
OWARI+0x54	687	146
OWARI+0x66	15	2

OWARI under scheme-1



command lines of download & kill added

More kill parameters added: "WsGA4@F6F", "ACDB", "AbAd", "iaGv", "902i13", "19ju3d"

kill parameters were added

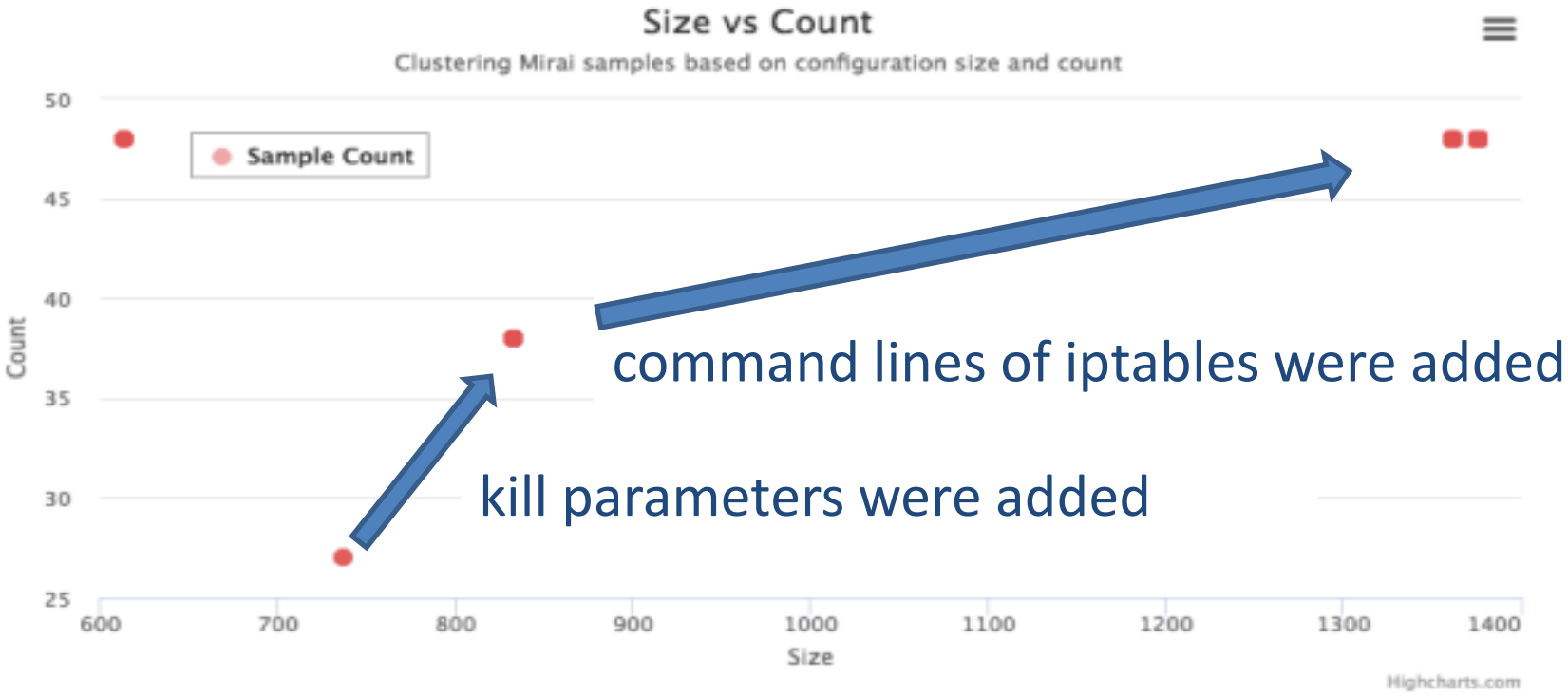
OWARI under scheme-4

- **15**: {0-atk_udp1, 1-atk_udp_vse1, 2-atk_tcp_syn1, 3-atk_tcp_ack1, 4-atk_gre1, 5-atk_gre1, 6-atk_std_or_udp, 7-atk_std_or_udp, 8-atk_tcp_stomp_or_xmas1}
- **642**: {0-atk_udp1, 1-atk_udp_vse1, 2-atk_udp_dns, 3-atk_tcp_syn1, 4-atk_tcp_ack1, 5-atk_tcp_stomp_or_xmas1, 6-atk_gre1, 7-atk_gre1, 8-atk_std_or_udp, 9-atk_std_or_udp, 10-atk_tcp_stomp_or_xmas1}

Outline

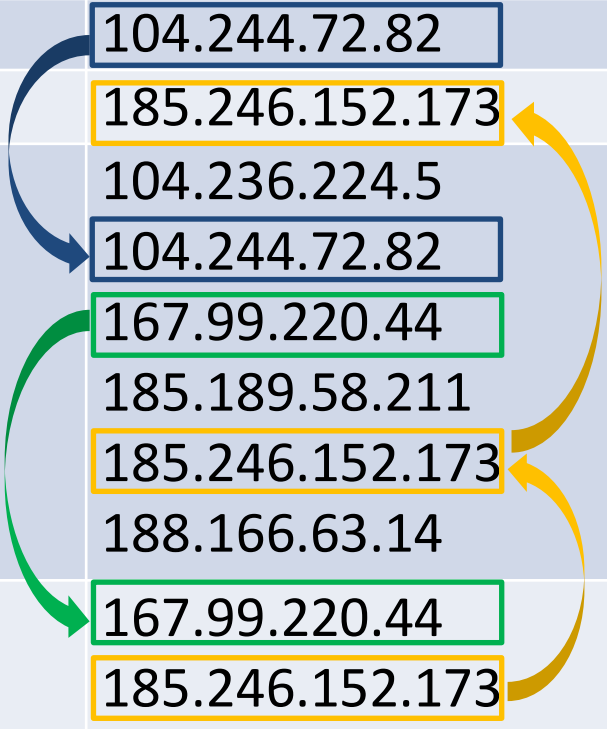
- Background
- Data and methodology
- Analysis of typical branches
  - MASUTA
  - OWARI
  - WICKED

WICKED under scheme-1



More details

(size, count)	Samples	CNC
(48, 614)	15	104.244.72.82
(27, 737)	4	185.246.152.173
(38, 833)	54	104.236.224.5 104.244.72.82 167.99.220.44 185.189.58.211 185.246.152.173 188.166.63.14
(48, 1362) (48, 1376)	55	167.99.220.44 185.246.152.173



WICKED on Scheme-4

- **4** combinations are obtained

```
 4  {0-atk_udp_or_gre2, 1-atk_udp_vse1, 2-atk_udp_dns, 3-atk_tcp_syn10, 6-atk_
 7  {0-atk_udp_or_gre2, 1-atk_udp_vse1, 2-atk_udp_dns, 3-atk_tcp_syn5, 6-atk_g
33  {0-atk_udp_or_gre2, 1-atk_udp_vse1, 2-atk_udp_dns, 3-atk_tcp_syn10, 6-atk_
71  {0-atk_udp_or_gre2, 1-atk_udp_vse1, 2-atk_udp_dns, 3-atk_tcp_syn5, 6-atk_g
```

Summary

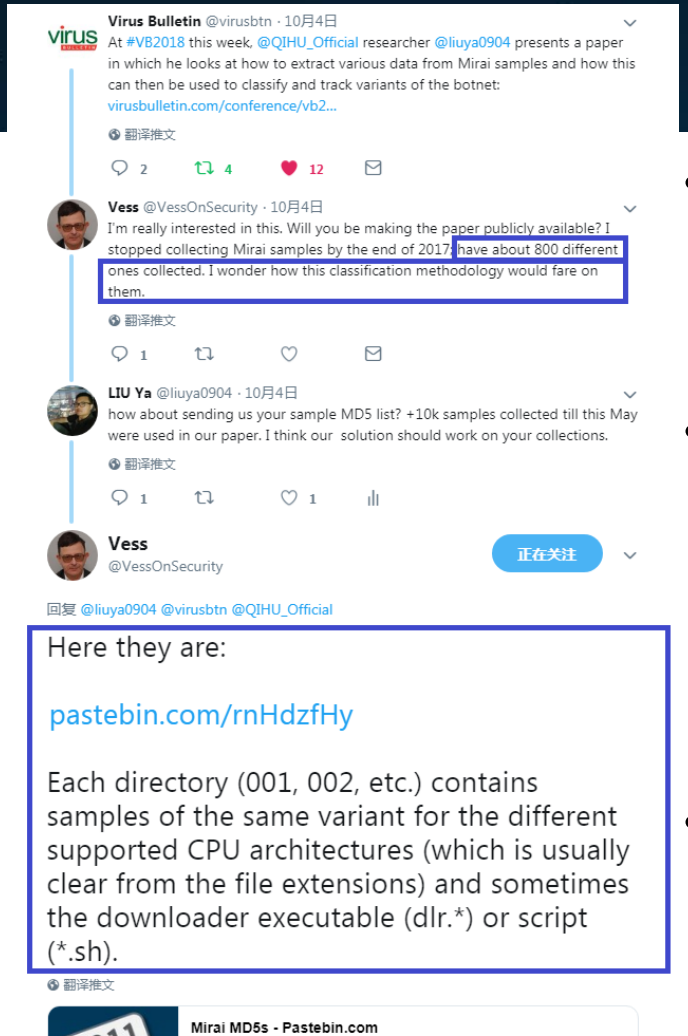
- Current Mirai variants and classifications were discussed
- Solutions of extracting 2 kinds of classification data were introduced
- 4 classification schemes based on the extracted data were demonstrated
- 3 popular Mirai branches were investigated with the demonstrated data and methodology

Future work

- To keep a tight watch on new exploits used by emerging Mirai variants
- To design classification with fuzzy hashing techniques (e.g., SSDP) to make better use of sample configurations
- To improve attack method fingerprinting techniques by considering the default option values

Classifications on VessOnSecurity's samples

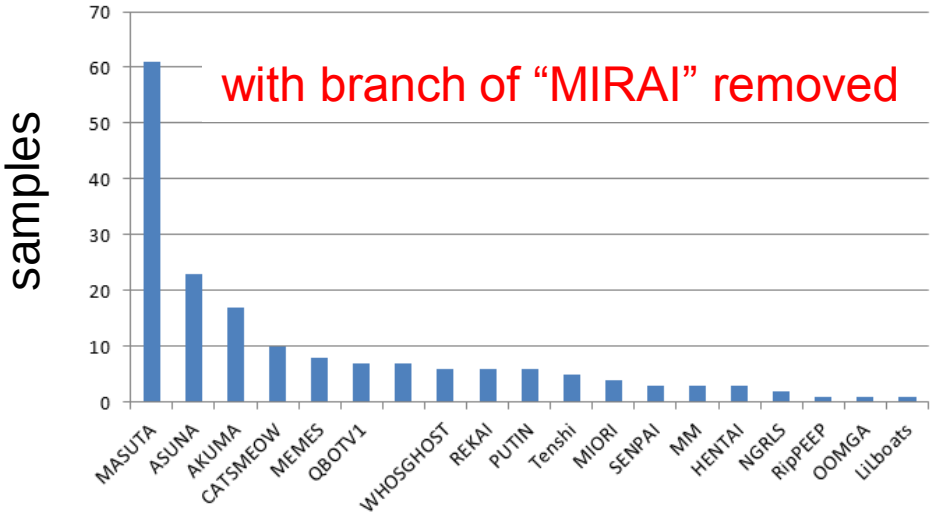
The left slides are later added to answer Vess's questions tweeted during VB2018 conference



- **7897** unique MD5s were found in Vess's sample set
  - https://pastebin.com/rnHdzfHy
- Only samples of x86 and ARM were considered
  - Because only those 2 kinds of samples were used in our talk for reasons of simplicity and efficiency
  - There should be no much accuracy loss due to Mirai's "one-source-to-multiple-processor" style of code compilation
- **1658** samples hit our talk's dataset
  - 520 x86 samples, 1132 ARM samples
  - https://pastebin.com/YCrpnmS4

20 branches found

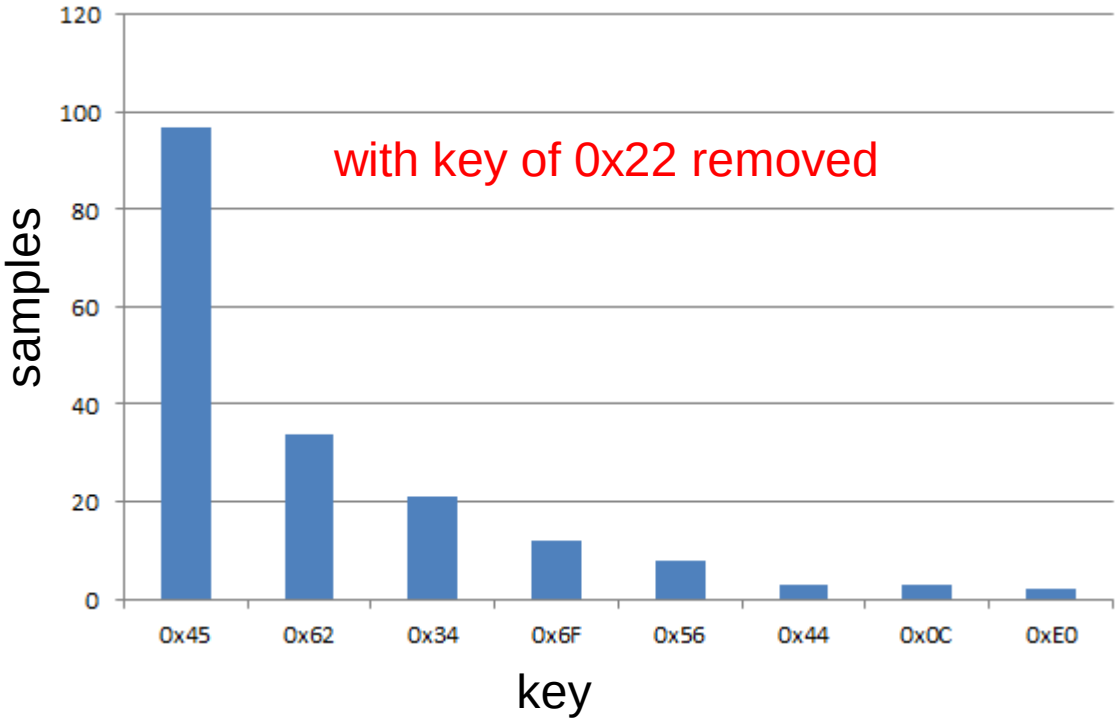
branch	samples
MIRAI	1484
MASUTA	61
ASUNA	23
AKUMA	17
CATSMEOW	10
MEMES	8
QBOTV1	7
NULL[*]	7
WHOSGHOST	6
REKAI	6
PUTIN	6
Tenshi	5
MIORI	4
SENPAI	3
MM	3
HENTAI	3
NGRLS	2
RipPEEP	1
OOMGA	1
LiLboats	1



[*] no branches found

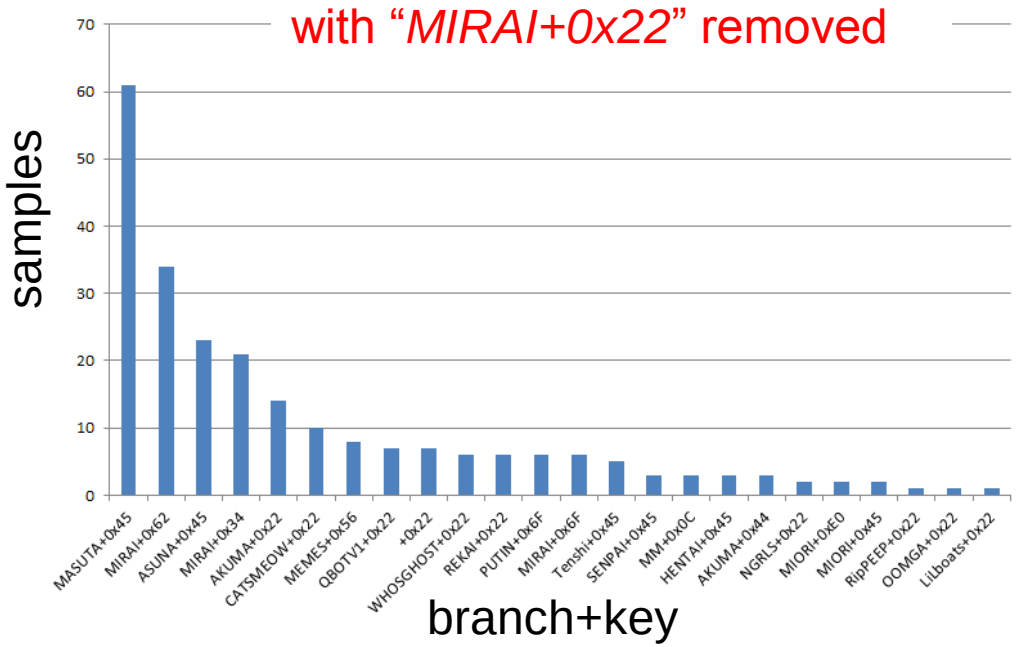
9 configuration encryption keys

key	samples
0x22	1478
0x45	97
0x62	34
0x34	21
0x6F	12
0x56	8
0x44	3
0x0C	3
0xE0	2

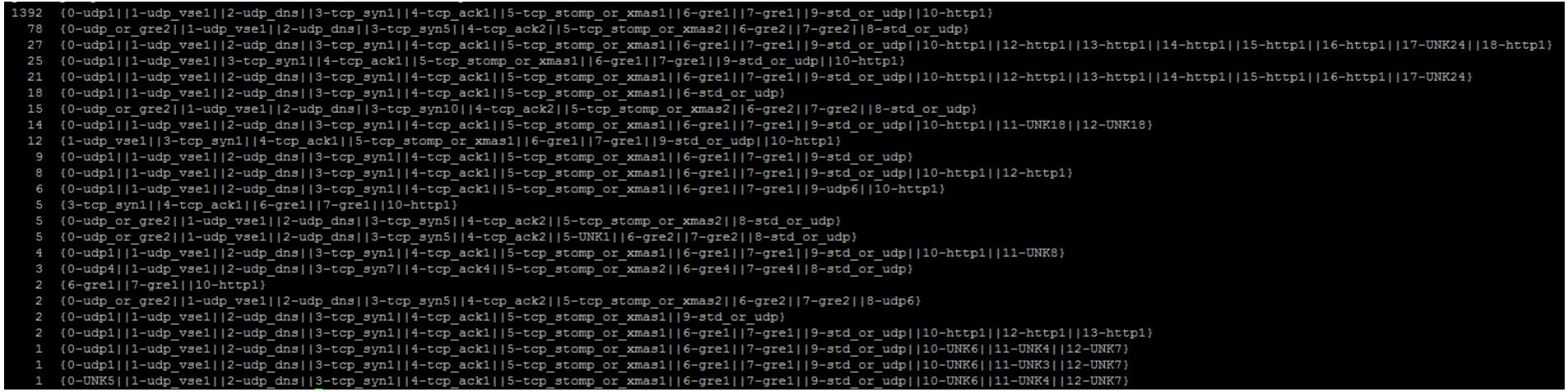


25 combinations of “branch+key”

branch+key	samples
MIRAI+0x22	1423
MASUTA+0x45	61
MIRAI+0x62	34
ASUNA+0x45	23
MIRAI+0x34	21
AKUMA+0x22	14
CATSMEOW+0x22	10
MEMES+0x56	8
QBOTV1+0x22	7
NUL+0x22	7
WHOSGHOST+0x22	6
REKAI+0x22	6
PUTIN+0x6F	6
MIRAI+0x6F	6
Tenshi+0x45	5
SENPAI+0x45	3
MM+0x0C	3
HENTAI+0x45	3
AKUMA+0x44	3
NGRLS+0x22	2
MIORI+0xE0	2
MIORI+0x45	2
RipPEEP+0x22	1
OOMGA+0x22	1
LiLboats+0x22	1



24 combinations of command coded attack FPs



```
1392  {0-udp1||1-udp_vse1||2-udp_dns||3-tcp_syn1||4-tcp_ack1||5-tcp_stomp_or_xmas1||6-gre1||7-gre1||9-std_or_udp||10-http1}
  78  {0-udp_or_gre2||1-udp_vse1||2-udp_dns||3-tcp_syn5||4-tcp_ack2||5-tcp_stomp_or_xmas2||6-gre2||7-gre2||8-std_or_udp}
  27  {0-udp1||1-udp_vse1||2-udp_dns||3-tcp_syn1||4-tcp_ack1||5-tcp_stomp_or_xmas1||6-gre1||7-gre1||9-std_or_udp||10-http1||12-http1||13-http1||14-http1||15-http1||16-http1||17-UNK24||18-http1}
  25  {0-udp1||1-udp_vse1||3-tcp_syn1||4-tcp_ack1||5-tcp_stomp_or_xmas1||6-gre1||7-gre1||9-std_or_udp||10-http1}
  21  {0-udp1||1-udp_vse1||2-udp_dns||3-tcp_syn1||4-tcp_ack1||5-tcp_stomp_or_xmas1||6-gre1||7-gre1||9-std_or_udp||10-http1||12-http1||13-http1||14-http1||15-http1||16-http1||17-UNK24}
  18  {0-udp1||1-udp_vse1||2-udp_dns||3-tcp_syn1||4-tcp_ack1||5-tcp_stomp_or_xmas1||6-std_or_udp}
  15  {0-udp_or_gre2||1-udp_vse1||2-udp_dns||3-tcp_syn10||4-tcp_ack2||5-tcp_stomp_or_xmas2||6-gre2||7-gre2||8-std_or_udp}
  14  {0-udp1||1-udp_vse1||2-udp_dns||3-tcp_syn1||4-tcp_ack1||5-tcp_stomp_or_xmas1||6-gre1||7-gre1||9-std_or_udp||10-http1||11-UNK18||12-UNK18}
  12  {1-udp_vse1||3-tcp_syn1||4-tcp_ack1||5-tcp_stomp_or_xmas1||6-gre1||7-gre1||9-std_or_udp||10-http1}
   9  {0-udp1||1-udp_vse1||2-udp_dns||3-tcp_syn1||4-tcp_ack1||5-tcp_stomp_or_xmas1||6-gre1||7-gre1||9-std_or_udp}
   8  {0-udp1||1-udp_vse1||2-udp_dns||3-tcp_syn1||4-tcp_ack1||5-tcp_stomp_or_xmas1||6-gre1||7-gre1||9-std_or_udp||10-http1||12-http1}
   6  {0-udp1||1-udp_vse1||2-udp_dns||3-tcp_syn1||4-tcp_ack1||5-tcp_stomp_or_xmas1||6-gre1||7-gre1||9-udp6||10-http1}
   5  {3-tcp_syn1||4-tcp_ack1||6-gre1||7-gre1||10-http1}
   5  {0-udp_or_gre2||1-udp_vse1||2-udp_dns||3-tcp_syn5||4-tcp_ack2||5-tcp_stomp_or_xmas2||8-std_or_udp}
   5  {0-udp_or_gre2||1-udp_vse1||2-udp_dns||3-tcp_syn5||4-tcp_ack2||5-UNK1||6-gre2||7-gre2||8-std_or_udp}
   4  {0-udp1||1-udp_vse1||2-udp_dns||3-tcp_syn1||4-tcp_ack1||5-tcp_stomp_or_xmas1||6-gre1||7-gre1||9-std_or_udp||10-http1||11-UNK8}
   3  {0-udp4||1-udp_vse1||2-udp_dns||3-tcp_syn7||4-tcp_ack4||5-tcp_stomp_or_xmas2||6-gre4||7-gre4||8-std_or_udp}
   2  {6-gre1||7-gre1||10-http1}
   2  {0-udp_or_gre2||1-udp_vse1||2-udp_dns||3-tcp_syn5||4-tcp_ack2||5-tcp_stomp_or_xmas2||6-gre2||7-gre2||8-udp6}
   2  {0-udp1||1-udp_vse1||2-udp_dns||3-tcp_syn1||4-tcp_ack1||5-tcp_stomp_or_xmas1||9-std_or_udp}
   2  {0-udp1||1-udp_vse1||2-udp_dns||3-tcp_syn1||4-tcp_ack1||5-tcp_stomp_or_xmas1||6-gre1||7-gre1||9-std_or_udp||10-http1||12-http1||13-http1}
   1  {0-udp1||1-udp_vse1||2-udp_dns||3-tcp_syn1||4-tcp_ack1||5-tcp_stomp_or_xmas1||6-gre1||7-gre1||9-std_or_udp||10-UNK6||11-UNK4||12-UNK7}
   1  {0-udp1||1-udp_vse1||2-udp_dns||3-tcp_syn1||4-tcp_ack1||5-tcp_stomp_or_xmas1||6-gre1||7-gre1||9-std_or_udp||10-UNK6||11-UNK3||12-UNK7}
   1  {0-UNK5||1-udp_vse1||2-udp_dns||3-tcp_syn1||4-tcp_ack1||5-tcp_stomp_or_xmas1||6-gre1||7-gre1||9-std_or_udp||10-UNK6||11-UNK4||12-UNK7}
```

- For consideration of limited space, prefixes of “atk_” are omitted
  - E.g., “atk_udp1 ” -> “udp1”

- UNKn stands for unknown attack fingerprints

Thank you

liuya@360.cn
wanghui3-s@360.cn