IIJ Internet Initiative Japan

mirai-toushi: Cross-Architecture Mirai Configuration Extractor Utilizing Standalone Ghidra Script

安全をあたりまえに
wizSafe

[TLP:CLEAR] Botconf2025

Shun Morishita, Satoshi Kobayashi, Eisei Hombu
Internet Initiative Japan Inc. (IIJ)

whoami

安全をあたりまえに wizSafe

- **Shun Morishita**
  - Internet Initiative Japan Inc. (IIJ)
    - Japanese ISP
  - Security analyst
    - Analyze security logs
    - Write analysis reports
    - Analyze malware
      - Primary focus on analyzing IoT malware to mitigate DDoS attacks

Agenda

- Motivation / Goal
- Mirai
- Existing Mirai Config Extractor
- mirai-toushi
- Evaluation
- Release
- Conclusion

Motivation / Goal

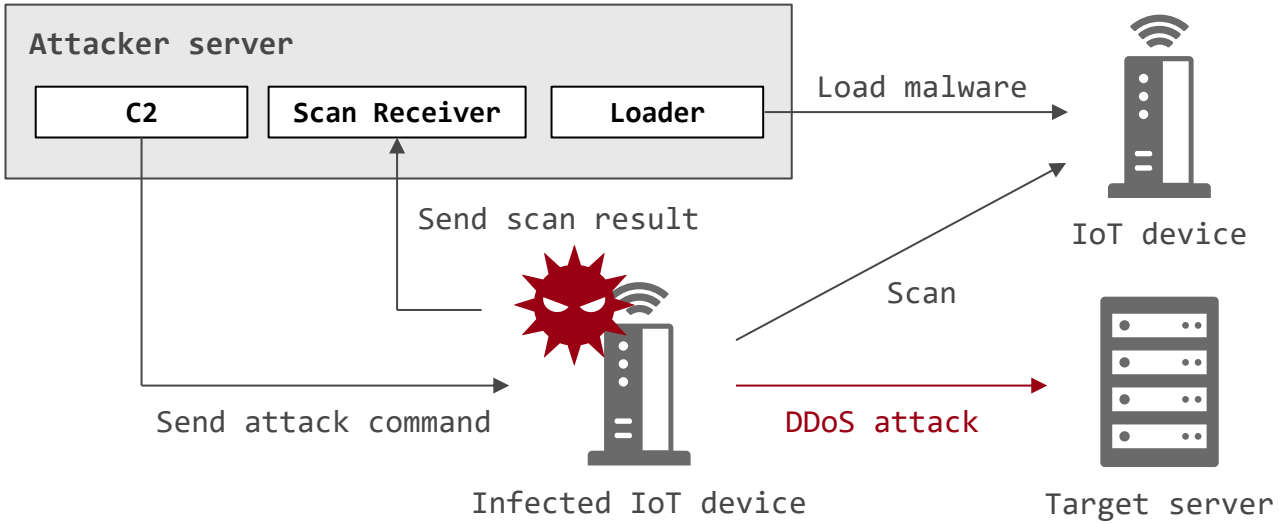
- Analyze IoT malware Mirai efficiently
  - Spend more time to analyze sophisticated Mirai variants
  - Spend less time to analyze original-based Mirai variants
    - The code is mostly the same, only the config is different

      -> Develop Mirai config extractor “**mirai-toushi**”

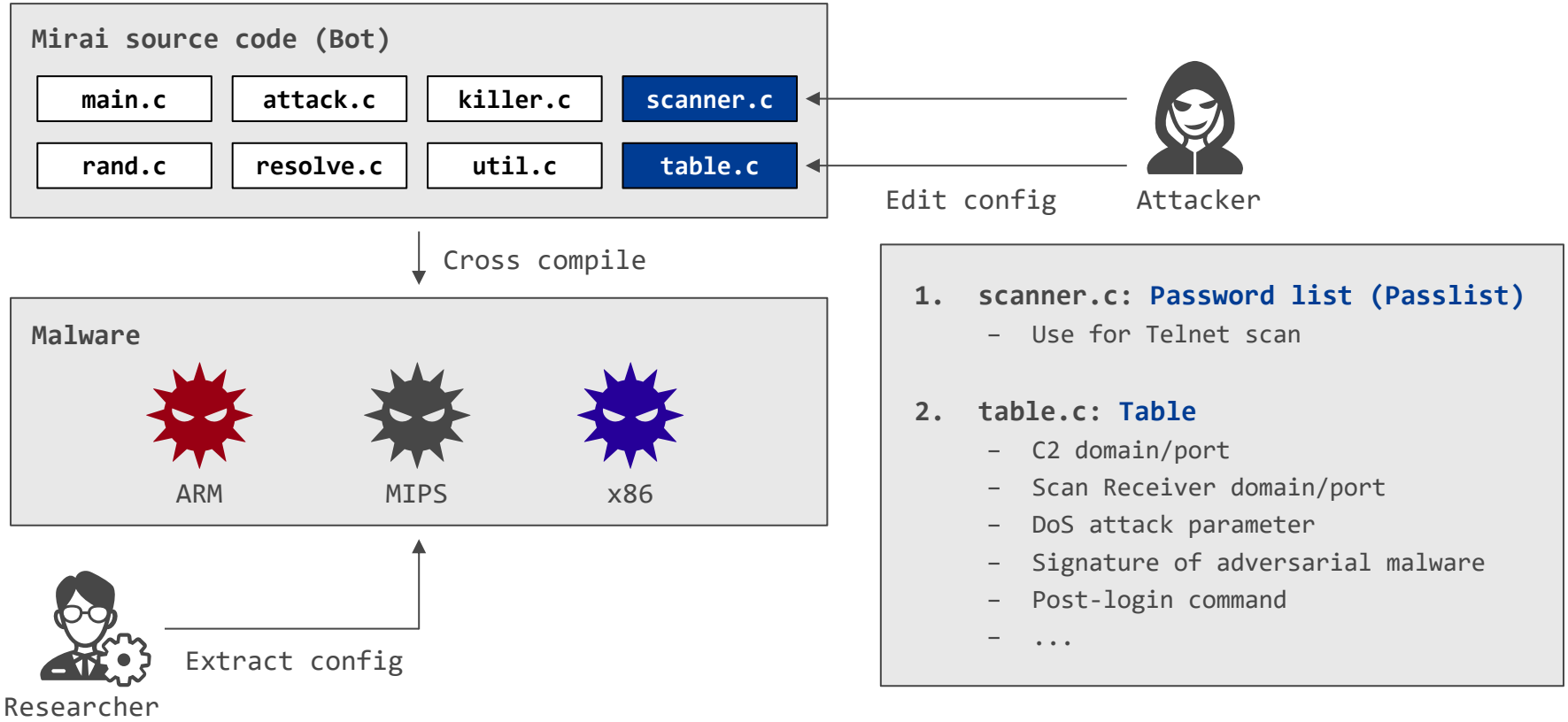
Existing extractor	Our extractor
• Specific-architecture	✓ Cross-architecture
• Partial config	✓ All config
• Partial automation	✓ Full automation

Mirai

- Mirai source code
  - Published in 2016, still now most IoT malware rely on it
  - Infect various IoT devices and launch DDoS attacks



Mirai config



1. scanner.c: **Password list (Passlist)**
   - Use for Telnet scan
2. table.c: **Table**
   - C2 domain/port
   - Scan Receiver domain/port
   - DoS attack parameter
   - Signature of adversarial malware
   - Post-login command
   - ...

Mirai config encryption

安全をあたりまえに wizSafe

Passlist registration	Table registration

`void scanner_init(void)`
`{`
`  // username, password, weight`
`  add_auth_entry("\x50\x4D\x4D\x56",`
`  "\x43\x46\x4F\x4B\x4C",`
`  8);`
	`void table_init(void)`
	`{`
	`  // id, data, data length`
	`  add_entry(TABLE_CNC_DOMAIN,`
	`  "\x47\x5A\x43\x4F\x52\x4E\x47\x0C\x41\x4D\x4F\x22",`
	`  12);`

- Encryption method
  - Split 4-byte XOR key and XOR 4 times = **1-byte XOR**
    - XOR key: **0xDEADBEEF**
      -> byte ⊕ 0xDE ⊕ 0xAD ⊕ 0xBE ⊕ 0xEF = byte ⊕ **0x22**

安全をあたりまえに
wizSafe

Mirai Config Extractor

Existing extractor

	Tool	Arch	Passlist	Table	Automation
decrypting-mirai-configuration-with-radare2	Radare2	x86		✓	
mirai_string_deobfuscation	Binary Ninja	ARM	✓		
miraicfg	Radare2	ARM, x86		✓	✓

1. decrypting-mirai-configuration-with-radare2
   - https://github.com/0xd3xt3r/blog-code/blob/master/decrypting-mirai-configuration-with-radare2
2. mirai_string_deobfuscation
   - https://github.com/mrphrazer/mirai_string_deobfuscation
3. miraicfg
   - https://github.com/FernandoDoming/miraicfg

Limitation1: supported architecture

	Tool	Arch	Passlist	Table	Automation
decrypting-mirai-configuration-with-radare2	Radare2	x86		✓	
mirai_string_deobfuscation	Binary Ninja	ARM	✓		
miraicfg	Radare2	ARM, x86		✓	✓

- Support 1 or 2 architectures

Limitation2: partial extraction

	Tool	Arch	Passlist	Table	Automation
decrypting-mirai-configuration-with-radare2	Radare2	x86		✓	
mirai_string_deobfuscation	Binary Ninja	ARM	✓		
miraicfg	Radare2	ARM, x86		✓	✓

- Extract passlist or table

Limitation3: automation

	Tool	Arch	Passlist	Table	Automation
decrypting-mirai-configuration-with-radare2	Radare2	x86		✓	
mirai_string_deobfuscation	Binary Ninja	ARM	✓		
miraicfg	Radare2	ARM, x86		✓	✓

- Few tools support automation
  - Need to manually determine XOR key and decrypting function

Limitation4: table usage

miraicfg example

```
"string_table": [
  "unstableishere\u0000",
  "example.com\u0000",      // C2?
  "example.net\u0000",      // Scan Receiver?
  "/proc/\u0000",
  "/exe\u0000",
  "/fd\u0000",
  "/mnt\u0000\u0004",
```

- Extraction is not enough to understand table usage
  - e.g., extracted domain is used for C2? Scan Receiver?

安全をあたりまえに
wizSafe

未来: Future

mirai-toushi

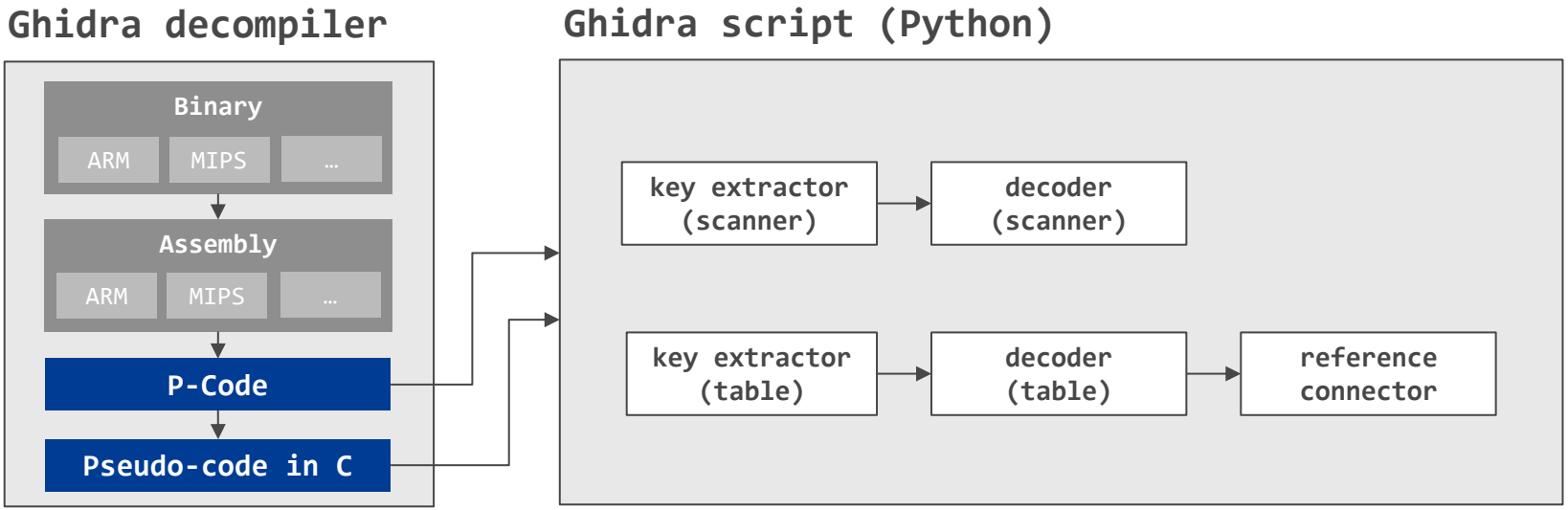
透視: Perspective
投資: Investment

	Tool	Arch	Passlist	Table	Automation
mirai-toushi	Ghidra	ARM, MC68000, MIPS, PowerPC, SPARC, SuperH4, x86, x86_64	✓	✓	✓

https://github.com/iij/mirai-toushi

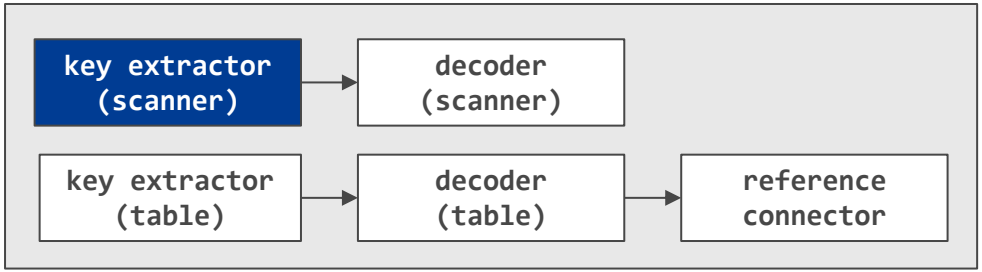
- ✓ Support 8 architectures
- ✓ Extract passlist and table
- ✓ Automation
- ✓ Determine table usage

Overview of mirai-toushi



- Utilize architecture-independent expressions
  - P-Code
  - Pseudo-code in C

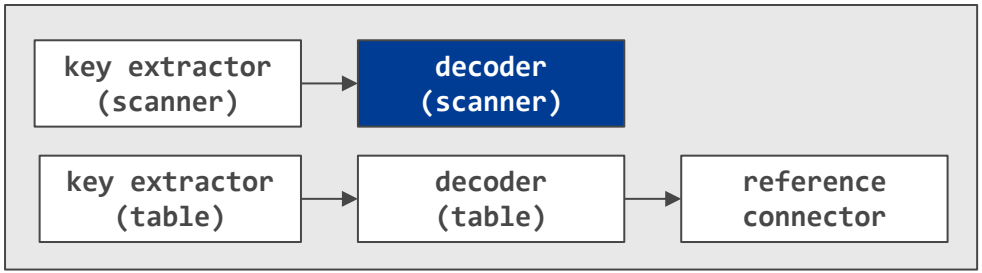
key extractor (scanner)



- Extract passlist XOR key
  - Use decompiled output
    - Identify the instruction which XORing 1-byte recursively

```
iVar4 = 0;
do {
  *(byte *)(iVar4 + (int)pvVar3) = *(byte *)(iVar4 + (int)pvVar3) ^ 0x22;
  iVar4 = iVar4 + 1;
} while (iVar2 != iVar4);
```

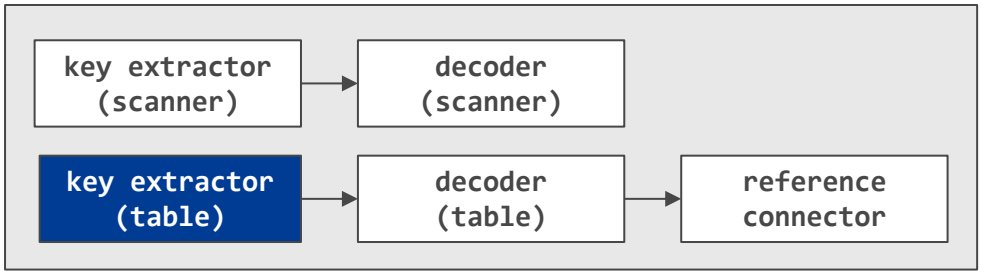
decoder (scanner)



- Decrypt passlist
  - Use decompiled output
    - Identify `add_auth_entry()`
    - Decrypt username (1st arg) and password (2nd arg)

```
add_auth_entry(&DAT_08054bcc,&DAT_08054bc5,10);
add_auth_entry(&DAT_08054bcc,"TKXZT",9);
add_auth_entry(&DAT_08054bcc,"CFOKL",8);
add_auth_entry("CFOKL","CFOKL",7);
add_auth_entry(&DAT_08054bcc,&DAT_08054bd1,6);
add_auth_entry(&DAT_08054bcc,"ZOJFKRA",5);
```

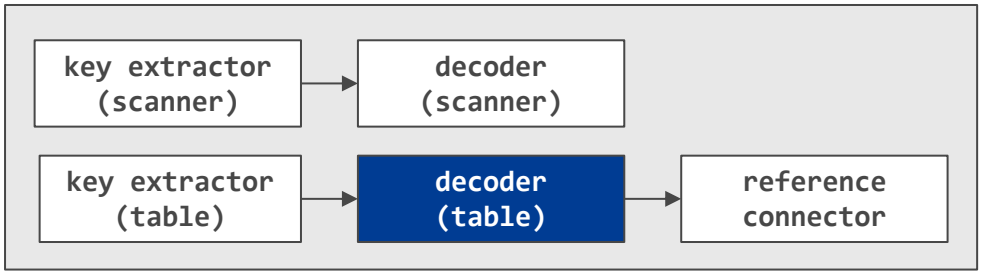
key extractor (table)



- Extract table XOR key
  - Use P-Code
    - Identify `INT_XOR` instructions executed 4 times

```
$U7800:1 = INT_XOR $U7800:1, BL
           ...
$U7800:1 = INT_XOR $U7800:1, BL
           ...
$U7800:1 = INT_XOR $U7800:1, BL
           ...
$U7800:1 = INT_XOR $U7800:1, BL
```

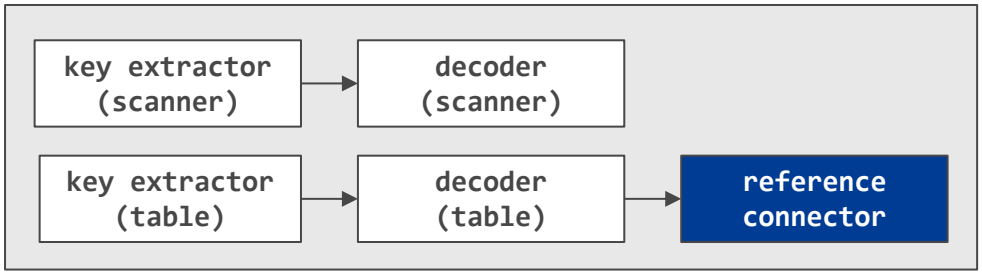
decoder (table)



- Decrypt table
  - Use decompiled output
    - Identify `add_entry()` / `util_memcpy()`
      - `add_entry()` is inlined due to compiler optimization
    - Decrypt table data (2nd arg)

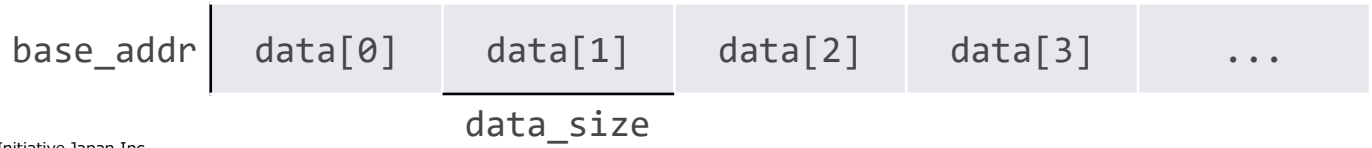
```
pvVar1 = malloc(0x1e);
util_memcpy(pvVar1,&DAT_08054d44,0x1e);
table._28_2_ = 0x1e;
table._24_4_ = pvVar1;
```

reference connector



- Determine where table is used
  - Use decompiled output
    - Identify `table_retrieve_val()` used to retrieve table data
    - `char *table_retrieve_val(int id, int *len);`
      - id: index of table (table is array type)

      `-> id = (data_addr - base_addr) / data_size`



Data size

	MC68000	Other 32-bit arch	x86_64
	6 bytes	8 bytes	16 bytes
0x00	Data Address	Data Address	Data Address
0x02			
0x04	Data Length	Data Length	
0x06		Padding	
0x08			Data Length
0x0A			Padding
0x0C			
0x0E			

Handle incorrect decompiled output

- Incorrect decompiled output
  - Decompiler misunderstands specific calling conventions
    - e.g., x86 uses stack to pass args, but some functions use register
  - Fix this problem using Ghidra script's **updateFunction()**
    - Define function variables correctly

Before

```
add_auth_entry();
add_auth_entry();
add_auth_entry();
add_auth_entry();
add_auth_entry();
add_auth_entry();
```

After

```
add_auth_entry(&DAT_08054bcc,&DAT_08054bc5,10);
add_auth_entry(&DAT_08054bcc,"TKXZT",9);
add_auth_entry(&DAT_08054bcc,"CFOKL",8);
add_auth_entry("CFOKL","CFOKL",7);
add_auth_entry(&DAT_08054bcc,&DAT_08054bd1,6);
add_auth_entry(&DAT_08054bcc,"ZOJFKRA",5);
```

Tuning

Collect **54** cross-compilers

- Collect from Mirai variant codes in GitHub

Build **370** verification samples

- Use original Mirai and 3 Mirai variant codes
  - MIRAI(0x22), Akiru(0xb4), SORA(0x54), WICKED(0x37)

Apply to mirai-toushi

- If it failed, we fixed the tool

364 samples

6 samples

due to Ghidra analysis error

Passlist output example

```
"add_auth_entry_func": {
  "name": "add_auth_entry",
  "entrypoint": "0804f8d0",
  "scanner_key": "0x22"                    // XOR key (1-byte)
},
"scanner_init_func": {
  "name": "scanner_init",
  "entrypoint": "0804fa20",
  "auth_tables_sha256": "0e60e37e94…",  // Passlist hash value
  "auth_tables_count": 23,
  "auth_tables": [
    {
      "user": "root",                      // Username
      "pass": "admin",                     // Password
      "weight": 8                          // Random selection weight
    },
```

- Sample: https://github.com/iij/mirai-toushi/tree/main/sample
- JSON Schema: https://github.com/iij/mirai-toushi/tree/main/jsonschema

Table output example

```
"table_lock_val_func": {
  "name": "table_lock_val",
  "entrypoint": "08050f80",
  "table_key": "0x22",                    // XOR key (1-byte)
  "table_original_key": "0xdeadbeef"      // Original XOR key (4-byte)
},
"table_init_func": {
  "name": "table_init",
  "entrypoint": "08051080",
  "tables_sha256": "5c4a784a20…",         // Table hash value
  "tables_count": 50,
  "tables_int_count": 2,
  "tables_str_count": 48,
  "tables": [
    {
      "id": 3,
      "type": "str",
      "str_data": "example.com",          // Decrypted data
      "table_addr": "080565d8",
      "refs": [
        {
          "func": "resolve_cnc_addr",     // Caller function
          "addr": "0804e552"              // Caller address
        }
```

安全をあたりまえに
wizSafe

Evaluation

Dataset

Dataset	System	Period	# Malware
IoTPOT Dataset D *	Honeypot	05/21/2020 - 12/31/2022	1,000
IoTPOT Dataset E *	Honeypot	01/01/2023 - 12/31/2023	1,000
IIJ-MALWARE	Honeypot, IPS	03/21/2024 - 05/31/2024	426

* IoTPOT, Yoshioka Lab, Yokohama National University, https://sec.ynu.codes/iot.

- Preprocess
  1. Unpack (UPX)
  2. Yara rule
     - Include passlist/table signatures encrypted by 1-byte XOR
       - Passlist: `root`, `admin`, `default`, `user`, `pass`
       - Table: `/proc`, `shell`, `enable`, `/bin/busybox`, `Mozilla`

Result: mirai-toushi vs miraicfg

		mirai-toushi			miraicfg
		Passlist	Table	Passlist||Table	Table
IoTPOT-D	1,000	862	808	958	339
IoTPOT-E	1,000	662	726	884	284
IIJ-MALWARE	426	117	209	247	50
Total	2,426	1,641	1,743	2,089	673

Result: table

		mirai-toushi			miraicfg
		Passlist	Table	Passlist||Table	Table
IoTPOT-D	1,000	862	808	958	339
IoTPOT-E	1,000	662	726	884	284
IIJ-MALWARE	426	117	209	247	50
Total	2,426	1,641	1,743	2,089	673

- mirai-toushi outperformed in all datasets
  - mirai-toushi supports 8 architectures (2,409 samples)
  - miraicfg supports 2 architectures (1,191 samples)

Result: IIJ-MALWARE

		mirai-toushi			miraicfg
		Passlist	Table	Passlist||Table	Table
IoTPOT-D	1,000	862	808	958	339
IoTPOT-E	1,000	662	726	884	284
IIJ-MALWARE	426	117	209	247	50
Total	2,426	1,641	1,743	2,089	673

- Fewer extractions in IIJ-MALWARE
  - Failed on the malware with non-O3 optimization level

Influence of compiler optimization level

- Most of leaked Mirai variant source codes use **O3**
  - We built verification samples with **O3**
  - In contrast, IIJ-MALWARE contains many **non-O3** samples

  -> This makes binaries different (especially at inlined function)

Not Inlined (non-O3)

```
void table_init(void)

{
  add_entry(3,&DAT_08058b08,0x1e);
```

Inlined (O3)

```
void table_init(void)

{
  undefined *puVar1;

  puVar1 = (undefined *)malloc(0x1e);
  util_memcpy(puVar1,&DAT_08054d44,0x1e);
  table._28_2_ = 0x1e;
  table._24_4_ = puVar1;
```

mirai-toushi result by architecture

	ARM	MC68000	MIPS	PowerPC	SPARC	SuperH4	x86	x86_64
# Malware	948	215	476	251	15	220	243	41
Passlist	69%	75%	68%	70%	20%	66%	68%	17%
Table	64%	83%	77%	79%	93%	72%	86%	24%
Passlist ||Table	86%	91%	86%	89%	93%	89%	92%	32%

- Extracted in all 8 architectures that we supported
  - avg. Passlist: 68%, Table: 72%, Passlist||Table: 86%
    - Few extractions at x86_64 (contain many IIJ-MALWARE)
  - Unsupported: AArch64 (8 samples), ARC (9 samples)

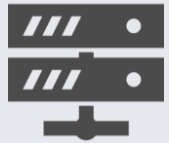
Difference between architectures

- **Packing**
  - All samples were not packed in specific architectures
    - ARC, MC68000, SPARC, and SuperH4
  - Due to **UPX**
    - UPX doesn’t support ELF file packing for these architectures

- **Stripping**
  - Many unstripped samples were ARM
    - 245 out of 282 unstripped samples were ARM
  - Due to **ARM cross-compiler**
    - Compilation error occurs when using strip command with specific ARM cross-compilers

Extracted passlist

Original Mirai	Real-World Mirai
Frequent • Router • IP camera • DVR	**Frequent** • Mobile router • DSL modem • ONU **Notable** • Dog feeder • Smart plug • Smart UPS



Extracted port

- # Port: 2,355
  - Higher than 1024
    - 2,236 (95%)
  - Top 10
    - 1,267 (54%)

- **Real-world malware tend to re-use ports**
  - 9 of top 10 ports are known Mirai variant ports

#	Port	Usage	Mirai variant
314	3912	SR	Cosmic-Mirai
312	1312	C2	Cosmic-Mirai
152	9555	SR	Condi-Boatnet
152	3778	C2	Condi-Boatnet
76	17661	?	?
56	1982	SR	Joker-Mirai
53	39284	SR	BeastMode V
52	34712	SR	Amari_Mirai_V2
52	17244	SR	DRACO 1.9 PRIVATE HYBRID
48	45	C2	Amari Mirai V2

* SR: Scan Receiver

Release

- We first uploaded experiment version

- We made several updates
  1. Extract configs from non-O3 malware
  2. Add new script “`parse_main.py`” to extract additional info
     - C2 in `resolve_cnc_addr()`
       - Some variants store C2 in `resolve_cnc_addr()`
     - DoS attack function
       - DoS attack functions are different depending on variants

parse_main.py output example

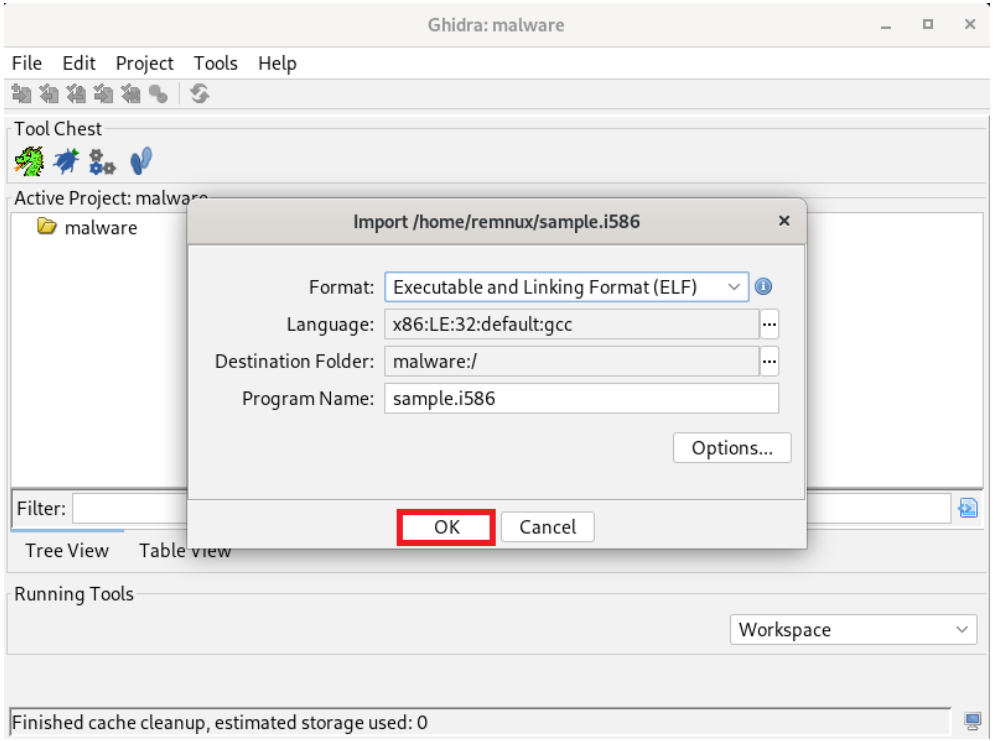
```
"main_func": {
  "name": "main",
  "entrypoint": "0804df60"
},
"resolve_cnc_addr_func": {
  "name": "resolve_cnc_addr",
  "entrypoint": "0804dc40",
  "cnc": "192.0.2.1"              // C2 in resolve_cnc_addr()
},
"attack_init_func": {
  "name": "attack_init",
  "entrypoint": "0804a630",
  "attacks_count": 5,
  "attacks": [
    {
      "vector": 0,
      "name": "attack_tcp_syn",   // DoS attack function
      "entrypoint": "0804b530"
    },
```

Usage

- If Ghidra is installed, mirai-toushi can be run without additional settings or libraries
  - Jython interpreter executed from Ghidra GUI
    - Ver. ~11.2 is Python interpreter
  - Headless analyzer executed from CUI

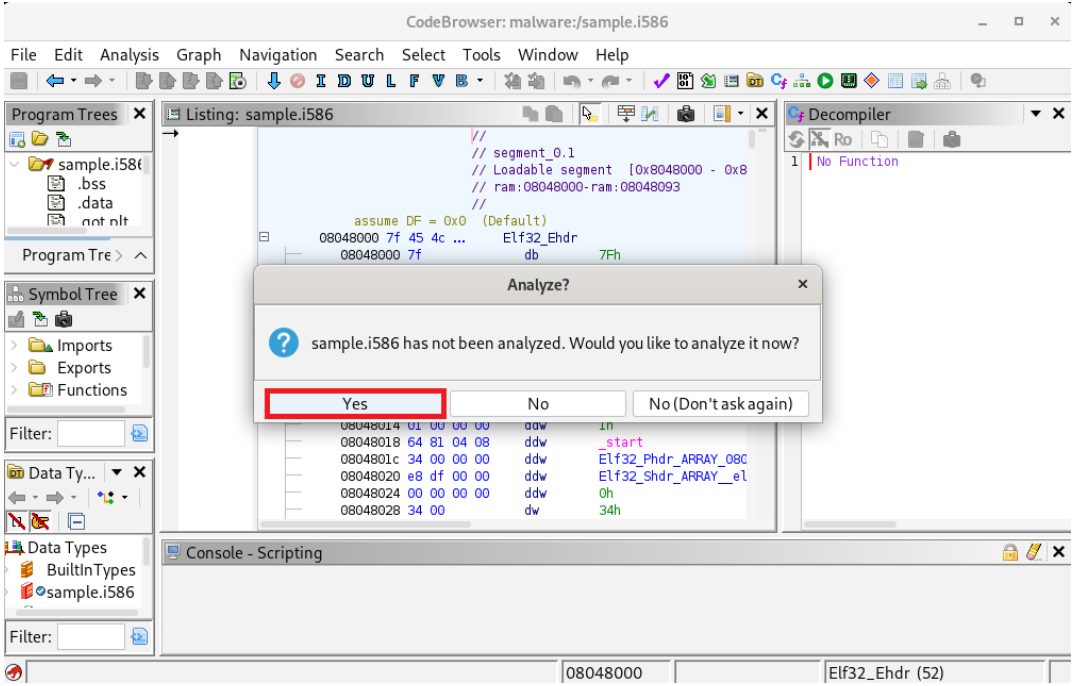
Usage: Ghidra GUI 1

- Start Ghidra GUI
- Create a project
- Import malware



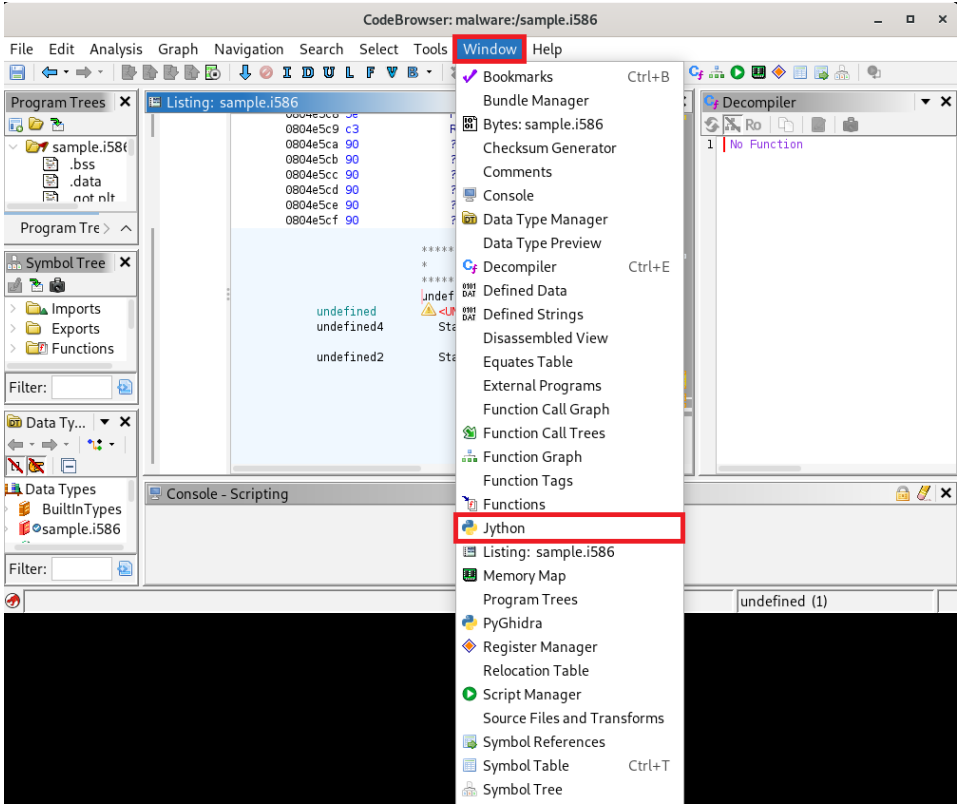
Usage: Ghidra GUI 2

- Run CodeBrowser
- Start initial analysis
  - Default option



Usage: Ghidra GUI 3

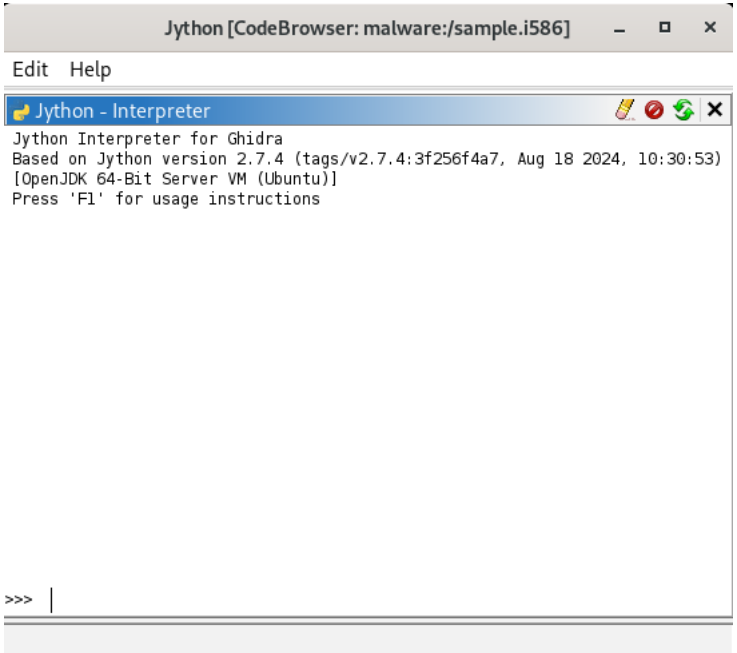
- Wait until analysis done
  - Progress bar shows at bottom right
- Start Jython interpreter
  - [Window] -> [Jython]
  - Ver. ~11.2: [Python]



Usage: Ghidra GUI 4

- Copy-paste Ghidra script
  - xor_scanner.py
  - xor_table.py
  - parse_main.py

- Result will be output as JSON



* Another usage:
  [Window] -> [Script Manager]
  - Need to add the script in `ghidra_scripts` folder

Usage: CUI

- Download mirai-toushi
  - `git clone https://github.com/iij/mirai-toushi`
- Run runner.sh
  - `chmod +x runner.sh`
  - `GHIDRA_INSTALL_DIR=<GHIDRA_DIR> ./runner.sh <ELF_FILE>`
    - At REMnux case, default Ghidra directory is `/opt/ghidra`
- Result will be output to `./output/<SHA256>/`

Usecase

1. Guess devices targeted by malware
   - Passlist

2. Use extracted config as IoC
   - Domain, IP address, and port

3. Detect new Mirai variants
   - Unknown SHA256 hash value

mirai-toushi limitation1

- Not effective against non-Mirai malware
  - Our methodology can be applied to other tools
- Take longer time to execute than existing tools
  - 50 seconds per malware
    - Execute at VM allocated 4GB memory and 1 core CPU (Host CPU: Intel Core i7-1185G7 @ 3.00GHz)

mirai-toushi limitation2

- Fail to extract configs if the encryption method, compiler, and optimization level are different
  - Sophisticated Mirai variants don’t use 1-byte XOR for table
    - e.g., Multi-byte XOR, ChaCha20, and RC4
    - We are trying to handle these cases using emulator…

Conclusion

- **mirai-toushi**: cross-architecture Mirai config extractor

- Apply to **2,426** real-world samples
  - Extract **1,641** passlists / **1,743** tables
  - Outperform the existing tool

- mirai-toushi is open-sourced

https://github.com/iij/mirai-toushi

wizSafe

安全をあたりまえに