FirmSolo: Enabling dynamic analysis of binary Linux-based IoT kernel modules

Ioannis Angelakopoulos, Gianluca Stringhini, and Manuel Egele

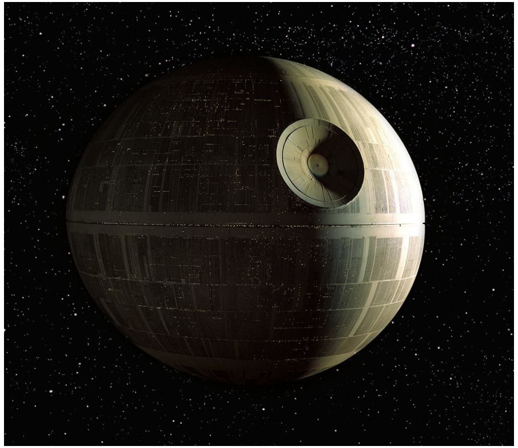
32nd Usenix Security Symposium

Date: 08/11/2023

SeclaBU

BOSTON UNIVERSITY

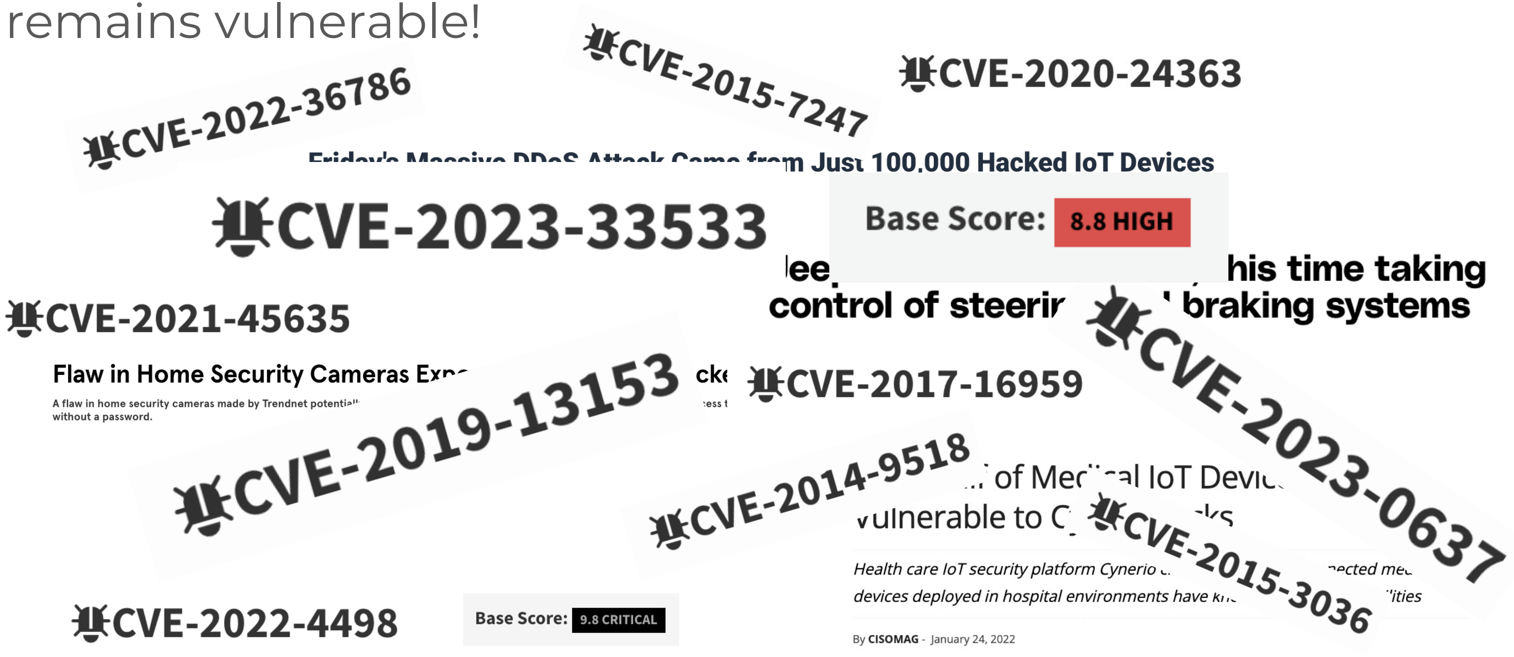
Internet of Things



Sources: https://www.amazon.com/
https://starwars.fandom.com/wiki/R2-D2
https://www.starwars.com/databank/death-star

IoT Security

IoT remains vulnerable!



CVE-2015-7247

CVE-2020-24363

CVE-2022-36786

Friday's Massive DDoS Attack Came from Just 100,000 Hacked IoT Devices

CVE-2023-33533

Base Score: 8.8 HIGH

lee ... his time taking control of steerir ... braking systems

CVE-2021-45635

Flaw in Home Security Cameras Exp ... cke

A flaw in home security cameras made by Trendnet potentiall ... cess t without a password.

CVE-2019-13153

CVE-2017-16959

CVE-2023-0637

CVE-2014-9518

of Medical IoT Devic ... vulnerable to C ... ks

CVE-2015-3036

Health care IoT security platform Cynerio ... nected me ... devices deployed in hospital environments have kn ... lities

By CISOMAG - January 24, 2022

CVE-2022-4498

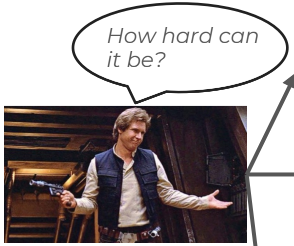
Base Score: 9.8 CRITICAL

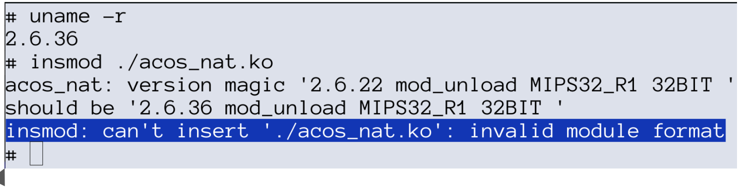
Linux-based IoT Firmware Analysis

- **Prior work primarily focused on user space!**
- Re-hosting
  - State-of-the-art emulators (e.g., QEMU) lack support for IoT hardware
  - Cannot emulate the firmware binary kernels!
- What about the IoT Linux loadable kernel modules (LKM)?
- What if LKMs only in binary form?
  - Frequent in the IoT domain!
  - Might contain vulnerabilities: *Kcode’s NetUSB.ko* (CVE-2015-3036)

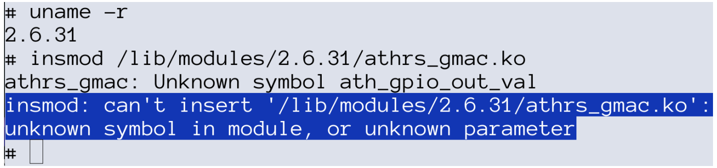
3 Key Challenges

Load IoT LKMs into QEMU supported kernels to dynamically analyze the LKMs

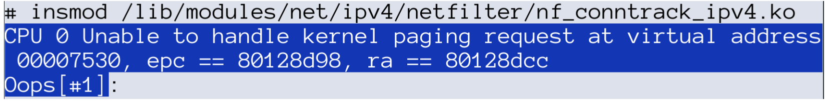




```
# uname -r
2.6.36
# insmod ./acos_nat.ko
acos_nat: version magic '2.6.22 mod_unload MIPS32_R1 32BIT '
should be '2.6.36 mod_unload MIPS32_R1 32BIT '
insmod: can't insert './acos_nat.ko': invalid module format
#
```



```
# uname -r
2.6.31
# insmod /lib/modules/2.6.31/athrs_gmac.ko
athrs_gmac: Unknown symbol ath_gpio_out_val
insmod: can't insert '/lib/modules/2.6.31/athrs_gmac.ko':
unknown symbol in module, or unknown parameter
#
```



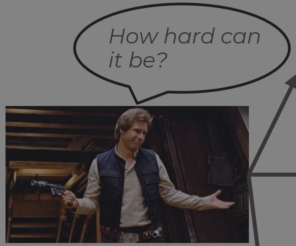
```
# insmod /lib/modules/net/ipv4/netfilter/nf_conntrack_ipv4.ko
CPU 0 Unable to handle kernel paging request at virtual address
 00007530, epc == 80128d98, ra == 80128dcc
Oops[#1]:
```

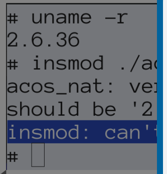
1. The kernel and kernel modules must be of the same version
2. External symbols (functions and data structures) the modules require must be provided:
   - Core kernel
   - Other modules loaded first (module dependencies)
3. The memory layout of data structures must be consistent between the kernel and the kernel modules
   - Misaligned data structure accesses!

3 Key Challenges

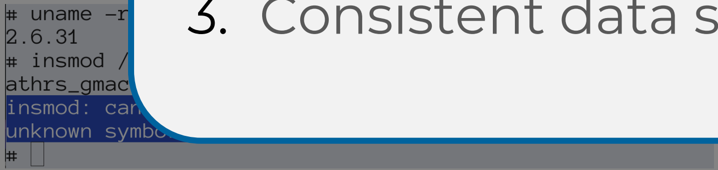
Load IoT LKMs into QEMU supported kernels to dynamically analyze the LKMs

1. Matching kernel version
2. Kernel symbol availability
3. Consistent data structure layouts

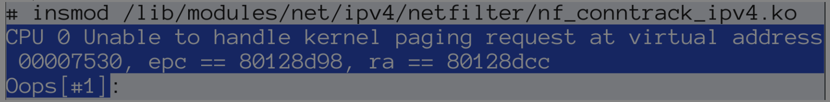




```
# uname -r
2.6.36
# insmod ./ac
acos_nat: ve
should be '2
insmod: can'
#
```



```
# uname -r
2.6.31
# insmod /
athrs_gmac
insmod: can
unknown symbo
#
```



```
# insmod /lib/modules/net/ipv4/netfilter/nf_conntrack_ipv4.ko
CPU 0 Unable to handle kernel paging request at virtual address
 00007530, epc == 80128d98, ra == 80128dcc
Oops[#1]:
```

l modules must be

ctions and data

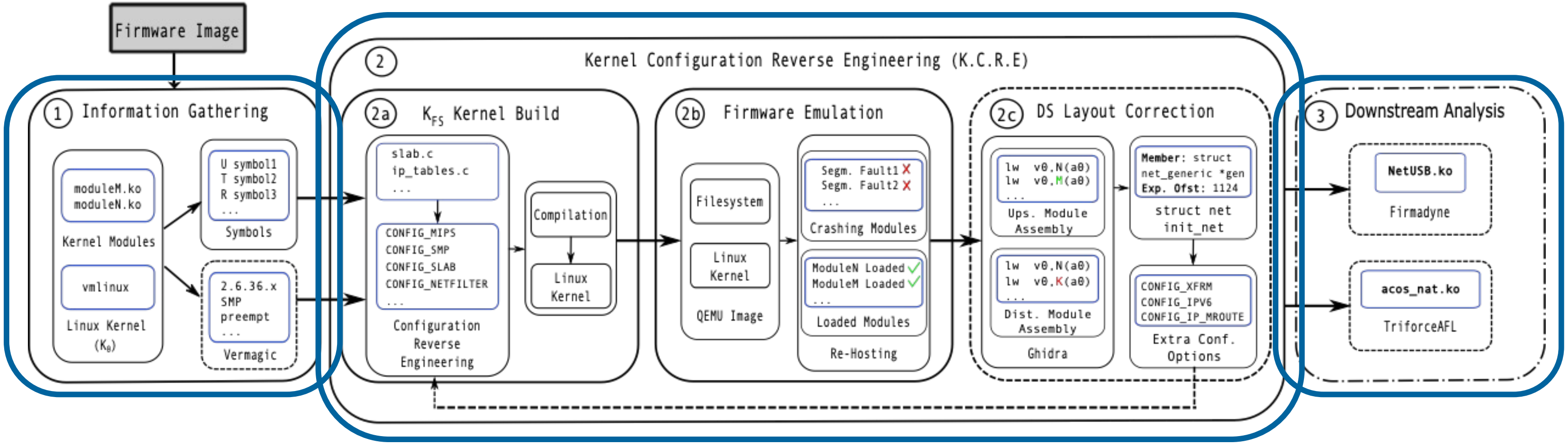
les require must be

ed first (module dependencies)

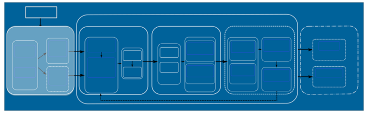
3. The memory layout of data structures must be consistent between the kernel and the kernel modules
    - Misaligned data structure accesses!

FirmSolo

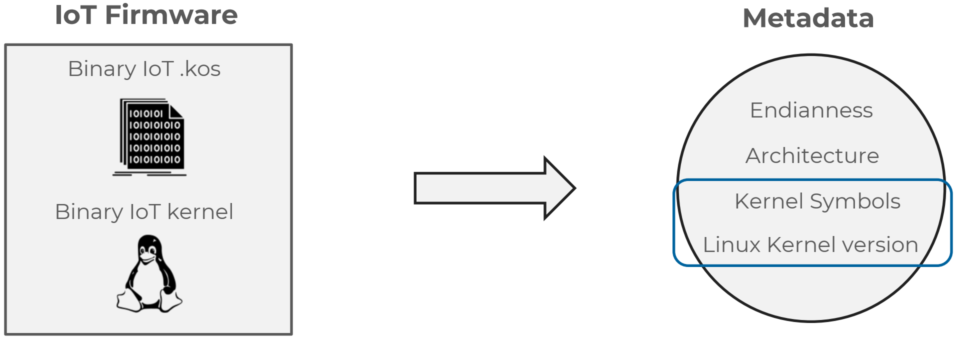
A framework to facilitate dynamic analysis of binary IoT kernel modules at scale



Information Gathering



- Gather metadata about the IoT firmware kernel
  - Used to compile a kernel that can load binary IoT kernel modules

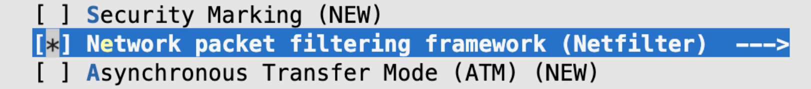


Challenges Revisited

Kernel symbol availability:

- Example:
  - *nf_register_hook*
- Defined in *net/netfilter/core.c*
- Guarded by *CONFIG_NETFILTER*

```
int nf_register_hook(struct nf_hook_ops *reg)
{
        struct nf_hook_ops *elem;
        int err;

        err = mutex_lock_interruptible(&nf_hook_mutex);
...
        list_add_rcu(&reg->list, elem->list.prev);
        mutex_unlock(&nf_hook_mutex);
        return 0;
}
EXPORT_SYMBOL(nf_register_hook);
```

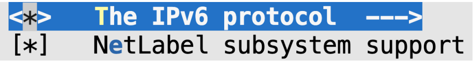


```
[ ] Security Marking (NEW)
[*] Network packet filtering framework (Netfilter)  --->
[ ] Asynchronous Transfer Mode (ATM) (NEW)
```

Consistent data structure layouts:

- Example:
  - *struct net*
- Represents network namespaces

```
struct net {
        atomic_t count;
        struct list_head list;
        struct list_head cleanup_list;
        struct list_head exit_list;
...
#if defined(CONFIG_IPV6)
        struct netns_ipv6 ipv6;
#endif
...
        struct sk_buff_head wext_nlevents;

        /* At offset &net + 0x324 */
        struct net_generic *gen;
};
```



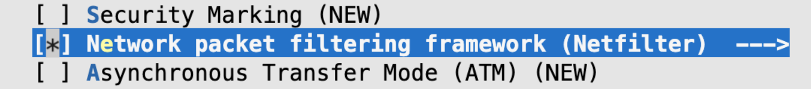
```
<*>   The IPv6 protocol  --->
[*]   NetLabel subsystem support
```

Challenges Revisited

Kernel symbol availability:

- Example:
  - *nf_register_hook*
- Defined in *net/netfilter/core.c*
- Guarded by *CONFIG_NETFILTER*

```
int nf_register_hook(struct nf_hook_ops *reg)
{
        struct nf_hook_ops *elem;
        int err;

        err = mutex_lock_interruptible(&nf_hook_mutex);
...
        list_add_rcu(&reg->list, elem->list.prev);
        mutex_unlock(&nf_hook_mutex);
        return 0;
}
EXPORT_SYMBOL(nf_register_hook);
```



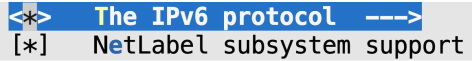
```
[ ] Security Marking (NEW)
[*] Network packet filtering framework (Netfilter)  --->
[ ] Asynchronous Transfer Mode (ATM) (NEW)
```

Consistent data structure layouts:

- Example:
  - *struct net*
- Represents network namespaces

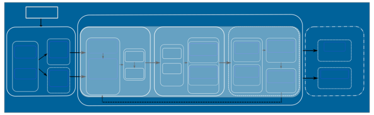
```
struct net {
        atomic_t count;
        struct list_head list;
        struct list_head cleanup_list;
        struct list_head exit_list;
...




        struct sk_buff_head wext_nlevents;

        /* At offset &net + 0x208 */
        struct net_generic *gen;
};
```

Crash!



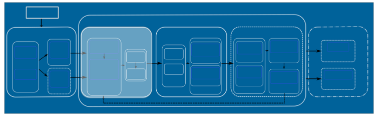
```
<*>   The IPv6 protocol  --->
[*]   NetLabel subsystem support
```

Kernel Configuration Reverse Engineering (K.C.R.E.)

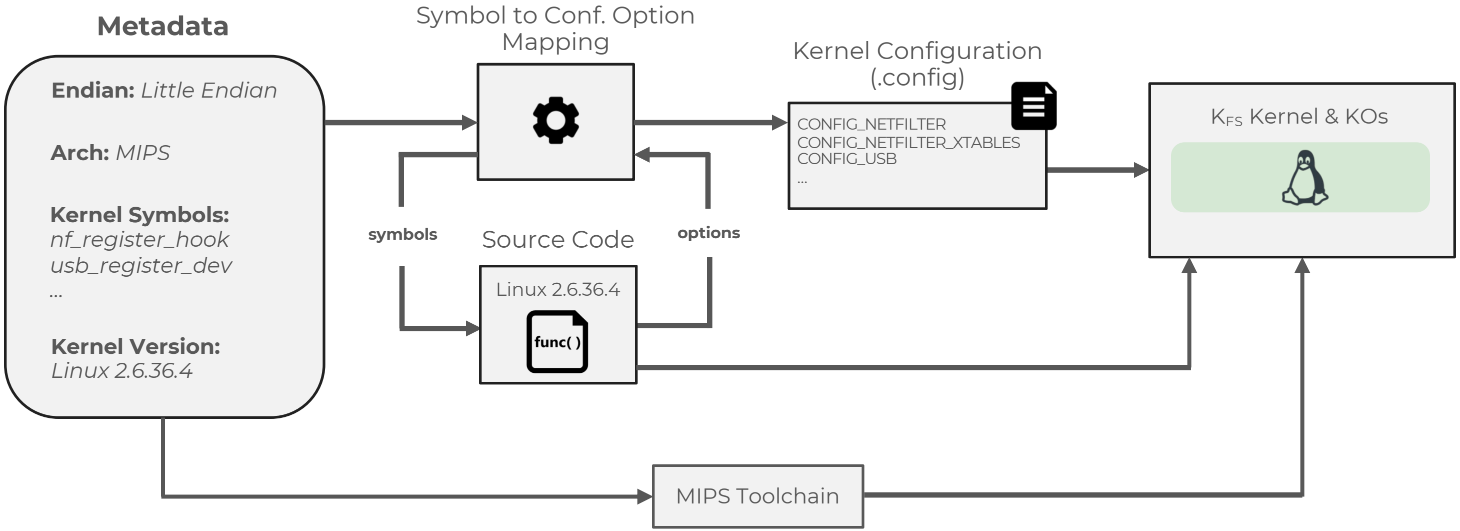


- Reverse engineer the IoT firmware kernel and compile a custom kernel capable of loading binary IoT kernel modules
- Consists of three steps:
  - K_{FS} Kernel build
  - Firmware Emulation
  - Data Structure Layout Correction (D.S.L.C.)

K_{FS} Kernel Build



- Build a kernel that can load binary IoT kernel modules
  - Reconstruct (approximately) the configuration of the IoT firmware kernel



Metadata

Endian: *Little Endian*

Arch: *MIPS*

Kernel Symbols:
nf_register_hook
usb_register_dev
...

Kernel Version:
Linux 2.6.36.4

Symbol to Conf. Option Mapping

symbols

options

Source Code

Linux 2.6.36.4

func( )

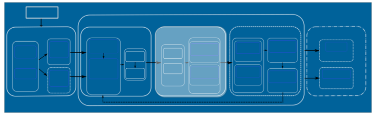
Kernel Configuration (.config)

CONFIG_NETFILTER
CONFIG_NETFILTER_XTABLES
CONFIG_USB
...

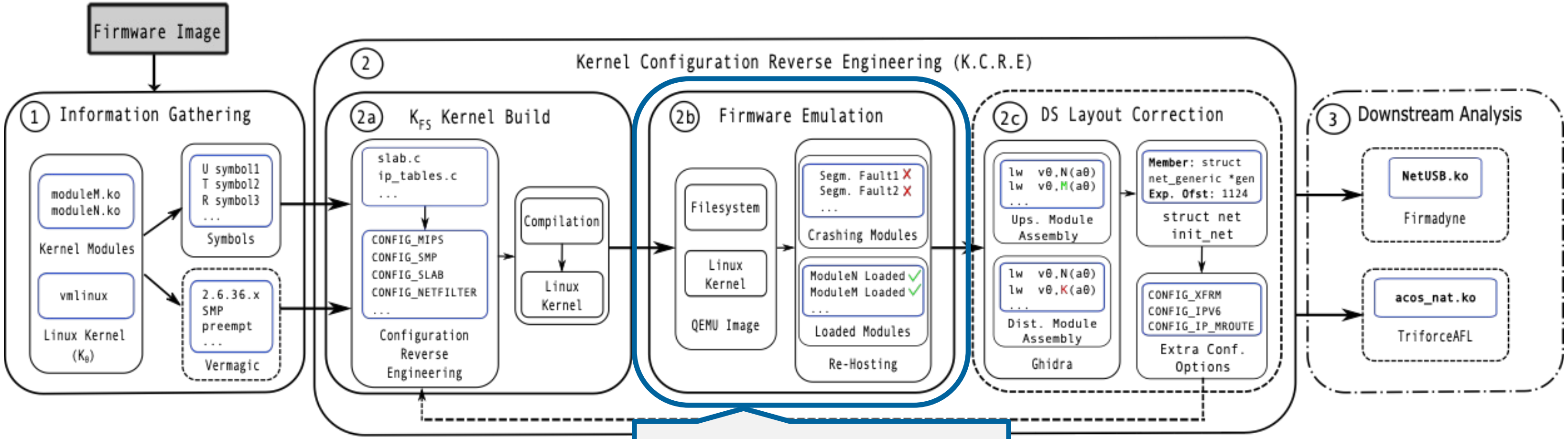
K_{FS} Kernel & KOs

MIPS Toolchain

Firmware Emulation

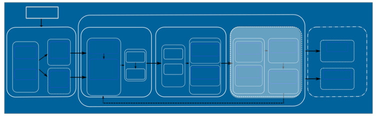


- Find which kernel modules can successfully load
  - Emulate the custom kernel K_{FS} and load the IoT kernel modules



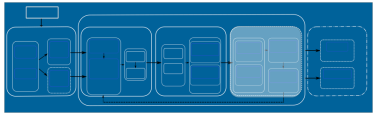
For details refer to the paper

Data Structure Layout Correction (D.S.L.C.)



- Address kernel module crashes related to misaligned data structure accesses
  - Re-introduce the offending kernel module back in the analysis
- Align data structure layouts
  - Find which configuration options when enabled/disabled fix the layout of the offending data structure

IoT vs Upstream kernel module



Example nf_conntrack_proto_gre.ko:

- IoT module Ghidra decompilation:

```
undefined4 proto_gre_net_init(int param_1)
{
bool bVar1;
undefined4 *puVar2;

bVar1 = true;
if (proto_gre_net_id != 0) {
        bVar1 = **(uint **)(param_1 + 0x324) < proto_gre_net_id;
}
...
return 0;
}
```

- IoT module Ghidra disassembly:

```
proto_gre_net_init:

00010058 lw v0,offset proto_gre_net_id &0xffff(v0)
0001005c lw v1,0x324(a0)
00010060 beq v0,zero,LAB_00010070
00010064 _li a0,0x1
00010068 lw a0,0x0(v1)
...
```

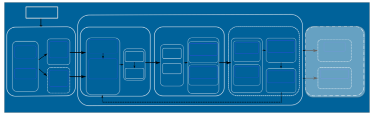
- Open-source module Ghidra decompilation:

```
int proto_gre_net_init(net *net)
{
bool bVar1;
undefined4 *puVar2;

bVar1 = true;
if (proto_gre_net_id != 0) {
        bVar1 = net->gen->len < (uint)proto_gre_net_id;
}
...
return 0;
}
```

- Open-source module Ghidra disassembly:

```
proto_gre_net_init:

00010090 lw v1,offset proto_gre_net_id(v0)
00010094 lw a0,0x208(a0)
00010098 beq v1,zero,LAB_000100a8
0001009c _li v0,0x1
000100a0 lw v0,0x0(a0)
...
```

Downstream Analysis



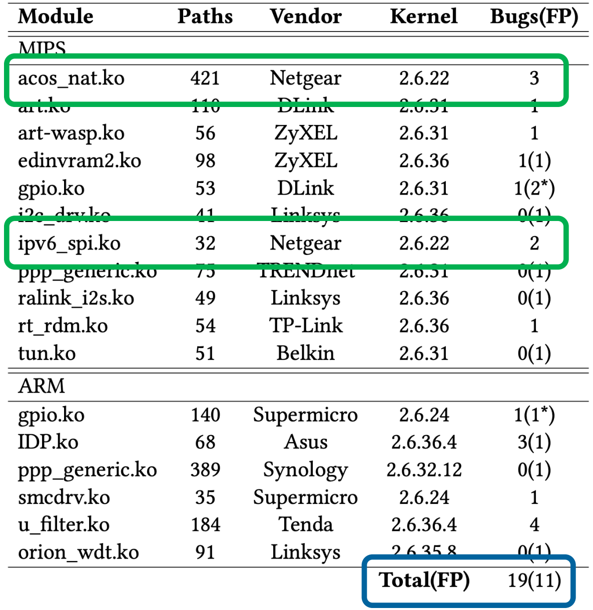
- Two examples of existing downstream analysis systems:
  - TriforceAFL [1]
    - Kernel module fuzzing via modules' IOCTL interface
  - Firmadyne [2]
    - Test against the bugs found by TriforceAFL
    - Test known exploits from ExploitDB [3]
- Others can be added!

[1] https://github.com/nccgroup/TriforceAFL
[2] Chen et al. Towards Automated Dynamic Analysis for Linux-based Embedded Firmware (NDSS 2016)
[3] https://www.exploit-db.com/

Sources: https://en.wikipedia.org/wiki/American_Fuzzy_Lop
https://mobile.twitter.com/exploitdb

Evaluation

- **Dataset:**
  - **1,470** firmware images
  - **56,688** binary kernel modules
- **FirmSolo:**
  - Loads **36,178** (64%) kernel modules
  - Previous work: 0% kernel modules
- **TriforceAFL:**
  - Fuzzed **75** LKMs with an IOCTL interface
  - Triggered **19** previously unknown bugs
  - Confirmed **5** bugs on a physical IoT device



Module	Paths	Vendor	Kernel	Bugs(FP)
MIPS				
acos_nat.ko	421	Netgear	2.6.22	3
art.ko	110	DLink	2.6.31	1
art-wasp.ko	56	ZyXEL	2.6.31	1
edinvram2.ko	98	ZyXEL	2.6.36	1(1)
gpio.ko	53	DLink	2.6.31	1(2*)
i2c_drv.ko	41	Linksys	2.6.36	0(1)
ipv6_spi.ko	32	Netgear	2.6.22	2
ppp_generic.ko	75	TRENDnet	2.6.31	0(1)
ralink_i2s.ko	49	Linksys	2.6.36	0(1)
rt_rdm.ko	54	TP-Link	2.6.36	1
tun.ko	51	Belkin	2.6.31	0(1)
ARM				
gpio.ko	140	Supermicro	2.6.24	1(1*)
IDP.ko	68	Asus	2.6.36.4	3(1)
ppp_generic.ko	389	Synology	2.6.32.12	0(1)
smcdrv.ko	35	Supermicro	2.6.24	1
u_filter.ko	184	Tenda	2.6.36.4	4
orion_wdt.ko	91	Linksys	2.6.35.8	0(1)
			Total(FP)	19(11)

Table: Fuzzer statistics and results for the vulnerable LKMs. The * indicates the False Positive might be an actual bug but requires hardware access to confirm.

Summary

- FirmSolo builds custom Linux kernels that can load binary IoT kernel modules
- Exposes these kernel modules to dynamic analysis
  - Two examples; TriforceAFL and Firmadyne
- Source available at: https://github.com/BUseclab/FirmSolo
- Contact us:
  - jaggel@bu.edu

Thank You!

Link to the paper!

ARTIFACT
EVALUATED
usenix
ASSOCIATION
AVAILABLE

ARTIFACT
EVALUATED
usenix
ASSOCIATION
FUNCTIONAL