



UNIVERSITÀ DEGLI STUDI DI CATANIA

DIPARTIMENTO DI MATEMATICA E INFORMATICA

Corso di Laurea Triennale in Informatica

Tesi di Laurea

NDS-CONSTRAIN'T: WEB SPOOFING CONTRO UNA CONSOLE NINTENDO DS

Relatore:
Chiar.mo Prof.
Giampaolo Bella

Candidato:
Federico Gaeta

Correlatore:
Dott. Cristian Daniele

ANNO ACCADEMICO 2021-2022

*Dedicata alla mia famiglia...
Federico*

Indice

Introduzione	1
1 Indirizzamento su internet	3
1.1 DNS	3
2 Connessione sicura su internet	5
2.1 HTTPS	5
2.2 SSL & TLS	6
3 Certificati	7
3.1 Formato X.509	7
3.2 Vincoli di base	8
3.3 Catena di fiducia	8
3.4 Firma Digitale	9
3.5 CSR	10
3.6 Emissione di un certificato	11
3.7 Verifica di un certificato	11
4 La falla	13
4.1 nds-constrain't	13
5 Funzionamento dell'attacco	15
5.1 Indirizzare la richiesta verso il server finto	15
5.2 Gestione della richiesta	16
5.3 Configurazione di nginx	18
5.4 Falsificazione del certificato	20
6 Descrizione dello script	23
6.1 checkOSVersion	23
6.2 checkCertFile	23
6.3 checkSudo	24
6.4 checkSources	24
6.5 getPackages	25
6.6 extractCertificateFiles	25

6.7	forgeCertificateFiles	26
6.8	buildOpensslNginx	27
6.9	createNginxNintendoSb	29
6.10	getDWCRRepo	29
6.11	configDnsmasq	30
6.12	addCronJob	31
6.13	endMessage	32
7	Esecuzione dell'attacco	33
7.1	Requisiti	33
7.2	Estrazione della NAND	34
7.3	Estrazione del file 00000011.app	35
7.4	Esecuzione dello script	36
7.5	Utilizzo del server	36
8	Conclusioni	39
	Bibliografia	41

Ringraziamenti

Ringrazio innanzitutto la mia famiglia, per essermi stata vicina ed avermi sostenuto durante tutto il mio percorso di studi. Un ringraziamento anche al mio relatore, il Professore Giampaolo Bella, per avermi dato la possibilità di presentare questo progetto, e al mio correlatore Cristian Daniele, per avermi seguito durante lo sviluppo della tesi. Un ringraziamento particolare va al mio amico e collega Domenico Blanco, per avermi consentito l'accesso alla sua VPS, senza la quale non avrei potuto fare tutti i test necessari allo sviluppo di tale tesi. Ringrazio infine i miei amici e colleghi, che mi hanno sempre supportato, anche nei momenti più difficili.

*Grazie,
Federico Gaeta*

Introduzione

Nintendo Co. Ltd. è un'azienda giapponese specializzata nella produzione di videogiochi e console ed è considerata una delle più grandi compagnie del Giappone. Fu fondata il 23 settembre 1889 da Yamauchi Fusajirō.

Una delle problematiche principali che ha dovuto affrontare nel corso degli anni è stata quella della pirateria e della sicurezza informatica. Tuttora esiste un **CFW**¹ per tutte le console che ha prodotto negli ultimi anni (DSi, 3DS, Wii, Wii U e Switch).

Ad oggi, la seconda console più venduta al mondo è la Nintendo DS², console portatile ideata e fabbricata da Nintendo, presentata nel 2004. Questa console ha un design a conchiglia e possiede due schermi LCD al suo interno, uno dei quali è uno schermo tattile. Essa offre anche un microfono incorporato e, a differenza delle console precedentemente prodotte da Nintendo, supporta la connettività wireless IEEE 802.11 (Wi-Fi). Essendo stata progettata molti anni fa, gli standard di sicurezza utilizzati per la connessione ad internet sono obsoleti. Basti pensare che, per connettersi online, essa necessita di una connessione Wi-Fi senza protezione o con protezione WEP³, protocollo ormai deprecato perché molto facile da bucare. Il 20 maggio 2014 il servizio *Nintendo Wi-Fi Connection* è stato cessato, quindi la console non può più connettersi per giocare online.

Questa tesi vuole dimostrare come sia possibile sfruttare una falla di sicurezza nella console Nintendo DS, mediante un bash script, per eseguire contro di essa un attacco di tipo *WEB Spoofing*, in modo da poterla far nuovamente connettere ad internet e giocare online.

¹CFW: Custom Firmware, modifica software del firmware originale di una console.

²DS: la sigla sta per "Dual Screen", ad indicare la peculiare caratteristica della console, ed è solitamente abbreviata come NDS o DS.

³WEP: Wired Equivalent Privacy, parte dello standard IEEE 802.11, ratificato nel 1999 e ormai deprecato.

1 | Indirizzamento su internet

Ogni dispositivo o server connesso ad internet è rappresentato da un indirizzo IP. Questo è un numero a 32 bit intervallato da un punto ogni 8 bit nel caso di IPv4, oppure a 128 bit intervallato ogni 16 bit dai due punti nel caso di IPv6. È stato creato un database distribuito, in cui ad ogni nome corrisponde un indirizzo IP. Il nome è costituito da una serie di parole intervallate tra loro da un punto, con il quale la macchina è conosciuta su internet. Quando ci si connette ad un sito, il browser deve risolvere il nome, ovvero deve fare una traduzione da url del sito ad IP. Per fare ciò ha bisogno del servizio **DNS**.

1.1 DNS

Il DNS¹ è un servizio presente in ogni macchina, che si occupa della risoluzione degli indirizzi url. Ha a disposizione due server DNS di riferimento, di livello più alto, a cui poter chiedere la risoluzione del nome. Questi, a loro volta, conoscono dei server DNS di livello più alto e, nel caso non riescano nella traduzione nome-IP, potranno richiedere la risoluzione ad altri server di livello superiore.

La richiesta viene fatta al servizio locale della macchina, il quale chiede la traduzione al servizio di livello superiore. Nel caso in cui questo non riesca a tradurre, si sale di livello fino a raggiungere quello più alto, ovvero la radice. Se neanche quest'ultimo sa risolvere, saprà indicare a chi chiedere per ottenere la risposta.

Il sistema DNS è fondamentale per il funzionamento di internet, dato che gli utenti e i servizi cercano le macchine soltanto in base al nome, quasi mai tramite il loro indirizzo. Se la traduzione non avviene, non si può raggiungere il sito desiderato.

¹DNS: Domain Name System.

2 | Connessione sicura su internet

Quando ci si connette ad internet, ad un sito o ad un servizio, c'è sempre il rischio che qualcuno possa mettersi in mezzo alla comunicazione per intercettare e spiare tutto il traffico. Questo un tempo era molto facile da fare. Oggi invece è stato ideato un protocollo per rendere sicura la navigazione online ed evitare ogni forma di intercettazione: **HTTPS**.

2.1 HTTPS

Il protocollo HTTPS è lo standard che permette la navigazione sicura su internet. Consiste nella comunicazione tramite il protocollo **HTTP**¹ all'interno di una connessione criptata dal protocollo **SSL**², o dal suo successore **TLS**³, tramite crittografia asimmetrica, fornendo un'autenticazione del sito web visitato, protezione della privacy e integrità dei dati scambiati tra le parti comunicanti. Viene fatta una cifratura bidirezionale delle comunicazioni tra un client e un server e vi è garanzia del fatto che si sta comunicando esattamente con il sito web voluto e non con un sito falso, creato da qualche utente malevolo. Garantisce inoltre che i contenuti delle comunicazioni tra l'utente e il sito web non possano essere intercettati o alterati da terzi.

Quando ci si connette ad un sito tramite protocollo HTTPS, il browser mostrerà l'icona di un lucchetto chiuso sulla barra dell'url, a sinistra. Questa indica che il protocollo HTTPS sta funzionando correttamente e nessuno sta cercando di intercettare il traffico. Al contrario, se qualcuno cercasse di mettersi in mezzo alla comunicazione o si entrasse in una pagina che non è quella originale, il lucchetto risulterebbe aperto e in questo modo si può essere in grado di capire se qualcosa non va.

Connettendosi invece ad un sito tramite HTTP, in assenza di una connessione criptata, non sarà presente l'icona del lucchetto e si correrà quindi il rischio che qualcuno spii per intero la navigazione su internet.

¹HTTP: Hypertext Transfer Protocol, protocollo a livello applicativo usato come principale sistema per la trasmissione d'informazioni sul web.

²SSL: Secure Sockets Layer, deprecato.

³TLS: Transport Layer Security, standard odierno.

2.2 SSL & TLS

SSL è un protocollo utilizzato per rendere sicura la navigazione su internet. Venne pubblicato nella sua versione 2.0 nel 1995 ma, dato che presentava alcune falle di sicurezza, venne aggiornato alla versione 3.0 nel 1996. Tuttavia, nonostante l'aggiornamento, presentava ancora problemi di sicurezza e quindi venne aggiornato a TLS, standard di sicurezza che viene usato oggi. Nel 2015 SSL venne deprecato, poiché considerato insicuro.

3 | Certificati

Per far sì che la crittografia asimmetrica funzioni, e quindi che si possa avere la certezza di stabilire una connessione al sito o server prescelto, è necessario utilizzare un sistema di certificazione che possa fare delle associazioni sicure e univoche tra un'identità e la sua chiave pubblica. Solo grazie ai certificati si può essere sicuri che una determinata chiave pubblica appartenga realmente al suo proprietario. Ogni certificato contiene informazioni sulla chiave pubblica, l'identità del proprietario e la firma digitale dell'entità che ne ha verificato i contenuti, che rappresenta quindi l'emittente del certificato. Tale entità è un ente terzo fidato ed è chiamata **CA**¹. Essa fornisce il certificato, lo autentica tramite firma digitale per evitarne la duplicazione e la falsificazione e fornisce anche la propria chiave pubblica per verificarlo. Chiunque riceva il certificato può verificarlo e quindi può assicurarsi che la chiave pubblica sia associata effettivamente al proprietario del certificato, tramite l'algoritmo di verifica della firma, le cui informazioni si trovano all'interno del certificato stesso.

3.1 Formato X.509

X.509 è il nome dello standard usato per definire il formato dei certificati a chiave pubblica forniti dalle autorità di certificazione. Nella seguente tabella sono rappresentati i campi che si trovano all'interno di un certificato:

V3	V2	V1	Versione
			Numero di serie
			ID dell'algoritmo di firma
			Nome della CA emittente
			Periodo di validità
		Nome del soggetto certificato	
		Informazioni sulla chiave pubblica del soggetto	
		ID univoco dell'emittente	
		ID univoco del soggetto	
		Estensioni	
		Firma	

¹CA: Certificate Authority, ente di certificazione fidato.

Il campo *Versione* indica la versione del certificato, può avere valore da 1 a 3 e da questo dipende la presenza o assenza di alcuni campi. Il campo *Numero di serie* è un numero intero non negativo e deve essere unico per ogni certificato emesso da una data CA. Il campo *ID dell'algoritmo di firma* contiene l'identificatore dell'algoritmo per l'algoritmo usato dalla CA per firmare il certificato. Il campo *Nome della CA emittente* identifica l'ente che ha firmato ed emesso il certificato e deve contenere un nome non vuoto. Il campo *Periodo di validità* è l'intervallo di tempo durante il quale la CA garantisce che manterrà l'informazione riguardante lo stato del certificato. Il campo è rappresentato da una sequenza di due date: la data in cui inizia il periodo di validità del certificato e la data in cui il periodo di validità del certificato finisce. Il campo *Nome del soggetto certificato* identifica l'entità associata alla chiave pubblica, che si trova all'interno del campo della chiave pubblica del soggetto. Il campo *Informazioni sulla chiave pubblica del soggetto* è usato per contenere la chiave pubblica e per identificare l'algoritmo in cui essa viene usata. I campi *ID univoco dell'emittente* e *ID univoco del soggetto* sono presenti solo nelle versioni 2 e 3 dei certificati e sono utilizzate per gestire la possibilità di riutilizzare il nome del soggetto o il nome dell'emittente. Il campo *Estensioni* può essere trovato solo nella versione 3 dei certificati, nei quali, se presente, rappresenta una sequenza di una o più estensioni del certificato e, inoltre, fornisce metodi per associare attributi aggiuntivi all'utente o alla chiave pubblica e per la gestione di rapporti tra le CA. Il campo *Firma*, presente in tutte le versioni dei certificati, contiene la firma digitale del certificato.

3.2 Vincoli di base

Tra le estensioni di un certificato SSL presenti nel campo *Estensioni*, ce n'è una chiamata **Vincoli di base**, rappresentata da una variabile booleana che può essere *True* o *False*. Questa indica se il soggetto del certificato è una CA o meno, e se quindi il certificato può essere usato per firmare altri certificati.

Tutti i certificati delle CA intermedie hanno la variabile di questa sezione con valore *True*, infatti essi possono essere usati per firmare altri certificati. Al contrario, i certificati dei dispositivi che si connettono ad internet, detti certificati foglia, come ad esempio quello della console Nintendo DS, hanno la variabile con valore *False*, infatti essi non possono essere usati per firmare altri certificati.

3.3 Catena di fiducia

Per far sì che il certificato sia valido, ci si deve assicurare che la CA firmataria non sia malevola. Bisogna quindi verificare anche l'associazione tra la CA e la sua chiave pubblica. La verifica avviene tramite un altro certificato, firmato con chiave pubblica di una CA di livello superiore. Si forma così una vera e propria gerarchia di certificazione, in cui, per ogni CA, viene certificata l'associazione con la rispettiva chiave pubblica

dalla CA di livello superiore. La gerarchia prende il nome di **catena di fiducia** e l'intera infrastruttura di scambio di chiavi prende il nome di **PKI**².

Ovviamente non si può salire all'infinito nella PKI, ci dovrà essere una CA che non abbia livelli ad essa superiori. Esiste infatti, all'interno della catena di fiducia, una CA radice, la cui chiave privata, utilizzata per firmare i certificati dei livelli sottostanti, corrisponde alla chiave pubblica ed è un ente di cui ci si deve necessariamente fidare.

Pertanto, quando si deve verificare la validità di un certificato, viene fatto un controllo a partire dal certificato foglia, fino al certificato radice, il quale, se valido, renderà validi tutti i certificati sottostanti, fino al certificato foglia. Ciò permetterà di accertarsi della validità di ogni certificato della catena, e quindi della validità del certificato foglia.

3.4 Firma Digitale

La firma digitale è uno strumento crittografico finalizzato alla firma di messaggi o alla verifica di tale firma, per fornire prova dell'autenticità di un messaggio o di un documento digitale. Essa garantisce:

- l'autenticazione del mittente: il ricevente può verificare l'identità di colui che ha firmato il messaggio;
- l'integrità del messaggio: il messaggio non può essere alterato dopo essere stato firmato;
- il non ripudio, il mittente non può negare di aver firmato il messaggio.

Per la gestione delle firme digitali occorre una coppia di chiavi:

- la chiave privata, conosciuta solo dal mittente, necessaria per la firma del messaggio
- la chiave pubblica del mittente, conosciuta da tutti, necessaria per la verifica della firma

Inoltre occorre un **algoritmo di hashing**³ per generare il **digest**⁴ del contenuto di ciò che si vuole mandare o verificare.

Il mittente, per applicare la firma, genera il digest del messaggio tramite un algoritmo di hashing, usa la sua chiave privata per cifrarlo e infine manda il messaggio o

²PKI: Public Key Infrastructure.

³Algoritmo di hashing: algoritmo che, preso in input un determinato flusso di bit di qualsiasi dimensione, restituisce in output un insieme di bit di dimensione fissata, associata univocamente al contenuto iniziale. L'algoritmo non è invertibile, quindi non è possibile ricostruire il flusso di bit iniziale a partire da quello che viene restituito in output.

⁴Digest: output di un algoritmo di hashing. Viene generato applicando l'algoritmo di hashing, passando come parametro il contenuto del messaggio da mandare.

il documento digitale insieme al digest cifrato, che ne rappresenta la firma digitale. Il ricevente, per verificare la firma, genera il digest del messaggio o del documento digitale ricevuto, usa la chiave pubblica del firmatario per decifrarne la firma digitale e confronta l'hash ottenuto dalla decifrazione con l'hash da lui calcolato precedentemente, a partire dal contenuto del messaggio o del documento digitale. Nel caso in cui i due hash siano identici, si avrà la certezza che le tre proprietà sopra descritte (autenticazione, integrità e non ripudio) siano garantite.

Tuttavia è stato scoperto che, in caso di utilizzo degli algoritmi *MD5* o *SHA-1*, è possibile generare due messaggi o due documenti digitali differenti aventi lo stesso hash, modificando manualmente porzioni di byte che li compongono, il che non garantisce le proprietà descritte precedentemente. Ciò viene definito *problema delle collisioni*.

3.5 CSR

In una PKI, la **CSR**⁵ è una richiesta inviata dal soggetto che vuole essere certificato ad una CA, al fine di ottenere un certificato digitale. Nella seguente tabella sono rappresentati i campi che si trovano al suo interno:

C	Paese
ST	Stato
L	Località
O	Organizzazione
CN	Nome comune
EMAIL	Indirizzo email

Tutti questi campi, che contengono le informazioni del soggetto che sta richiedendo l'emissione di un certificato firmato, si trovano sotto la sezione *Nome soggetto*. Un campo importante è il *Nome comune*, in quanto contiene il dominio completo per cui si vuole ottenere il certificato. Oltre alla sezione prima nominata, la CSR è composta anche dalle seguenti sezioni: *Richiesta di certificato*, nella quale sono indicate il tipo e la versione del certificato che si vuole ottenere, *Informazioni chiave pubblica*, nella quale sono indicate la chiave pubblica, le informazioni che la riguardano e l'algoritmo in cui essa viene usata, *Firma*, nella quale sono presenti la firma digitale che garantisce la validità della CSR, l'identificativo dell'algoritmo di firma utilizzato e i parametri della firma.

La firma digitale viene generata cifrando, tramite la chiave privata del soggetto richiedente, l'hash calcolato dalle informazioni in chiaro della CSR e infine viene aggiunta in fondo alla CSR, prima che questa venga mandata alla CA.

⁵CSR: Certificate Signing Request, richiesta fatta ad una CA per ottenere un certificato firmato da essa.

3.6 Emissione di un certificato

Quando un soggetto vuole ricevere un certificato da una CA, che ne certifichi l'associazione con la sua chiave pubblica, deve generare una coppia di chiavi, una chiave privata che conosce solo lui e una chiave pubblica che conoscono tutti, infine anche un file CSR. Questo, al suo interno, conterrà la chiave pubblica generata e la firma digitale ottenuta tramite la chiave privata del soggetto. Quando la CA riceve la richiesta, essa verifica la firma tramite la chiave pubblica del soggetto presente nella CSR.

Se la richiesta e la verifica della firma vanno a buon fine, la CA invierà al soggetto un certificato digitale, firmato tramite la propria chiave privata.

3.7 Verifica di un certificato

Quando un soggetto vuole verificare la firma digitale di un certificato, per accertarsi della sua validità, deve decifrarla tramite la chiave pubblica della CA firmataria del certificato. Il risultato deve essere confrontato con l'hash generato dai dati in chiaro presenti nel certificato. Se i due valori sono uguali, allora si potrà essere sicuri che il certificato è valido e che la chiave pubblica presente all'interno del certificato sia effettivamente associata al proprietario.

Ovviamente, per essere sicuri che la chiave pubblica della CA firmataria sia effettivamente associata ad essa, bisogna verificare anche il certificato di tale chiave. Per far ciò, bisogna eseguire varie verifiche, salendo fino al certificato radice. Se tutta la catena viene verificata, allora il certificato può essere considerato valido.

4 | La falla

Essendo la console Nintendo DS risalente al 2004, le comunicazioni online avvengono principalmente tramite protocollo HTTP trasportato all'interno di pacchetti crittografati tramite SSL. Quando la console si connette ad un server ufficiale Nintendo, questo le presenta un certificato SSL contenente la propria chiave pubblica. La console la utilizzerà per criptare una chiave segreta che verrà utilizzato da entrambi per criptare tutte le comunicazioni successive.

I certificati SSL sono tutti firmati da un'autorità di certificazione Nintendo, in modo tale che la console sia sicura che il server al quale essa si sta connettendo sia affidabile. Quindi, se si provasse a far connettere la console ad un server privo di certificato firmato da una CA Nintendo fidata, questa rifiuterebbe la connessione. Perciò, dato che Nintendo ha chiuso tutti i server online per la Nintendo DS, questa teoricamente non può più connettersi ad internet.

La falla, che permette di fare *WEB Spoofing* contro la console e che quindi riesce a farla connettere nuovamente ad internet, non è dovuta al fatto che SSL sia ormai un protocollo insicuro e deprecato, ma è dovuta all'implementazione, fatta da Nintendo, del controllo della catena di fiducia da parte della console.

4.1 nds-constrain't

Se il server SSL unisce, in un unico certificato da mostrare alla console, il certificato della CA Nintendo intermedia al proprio certificato firmato dalla stessa, il client può seguire la catena di fiducia fino alla radice e quindi accettare il certificato.

Ed è proprio qui che risiede la falla: nell'implementazione fatta da Nintendo. Infatti la console non controlla se effettivamente il firmatario del certificato può firmare o meno altri certificati, ovvero se questo è una CA intermedia o meno, come viene indicato nella sua sezione *Vincoli di base*. Essa ignora completamente questa sezione e quindi, avendo a disposizione un qualsiasi certificato firmato dalla CA di Nintendo, anche un certificato che non sia di una CA Nintendo intermedia, e la sua chiave privata usata per firmarlo, li si possono utilizzare per firmare altri certificati che la console accetterebbe senza problemi.

Il nome dato all'exploit deriva appunto della denominazione della sezione *Vincoli di base*, la cui traduzione inglese è **Basic constraints**. Dato che Nintendo non

implementa correttamente la certificazione SSL nelle console Nintendo DS, ignorando completamente quel campo, alla falla è stato dato il nome **nds-constrain't**.

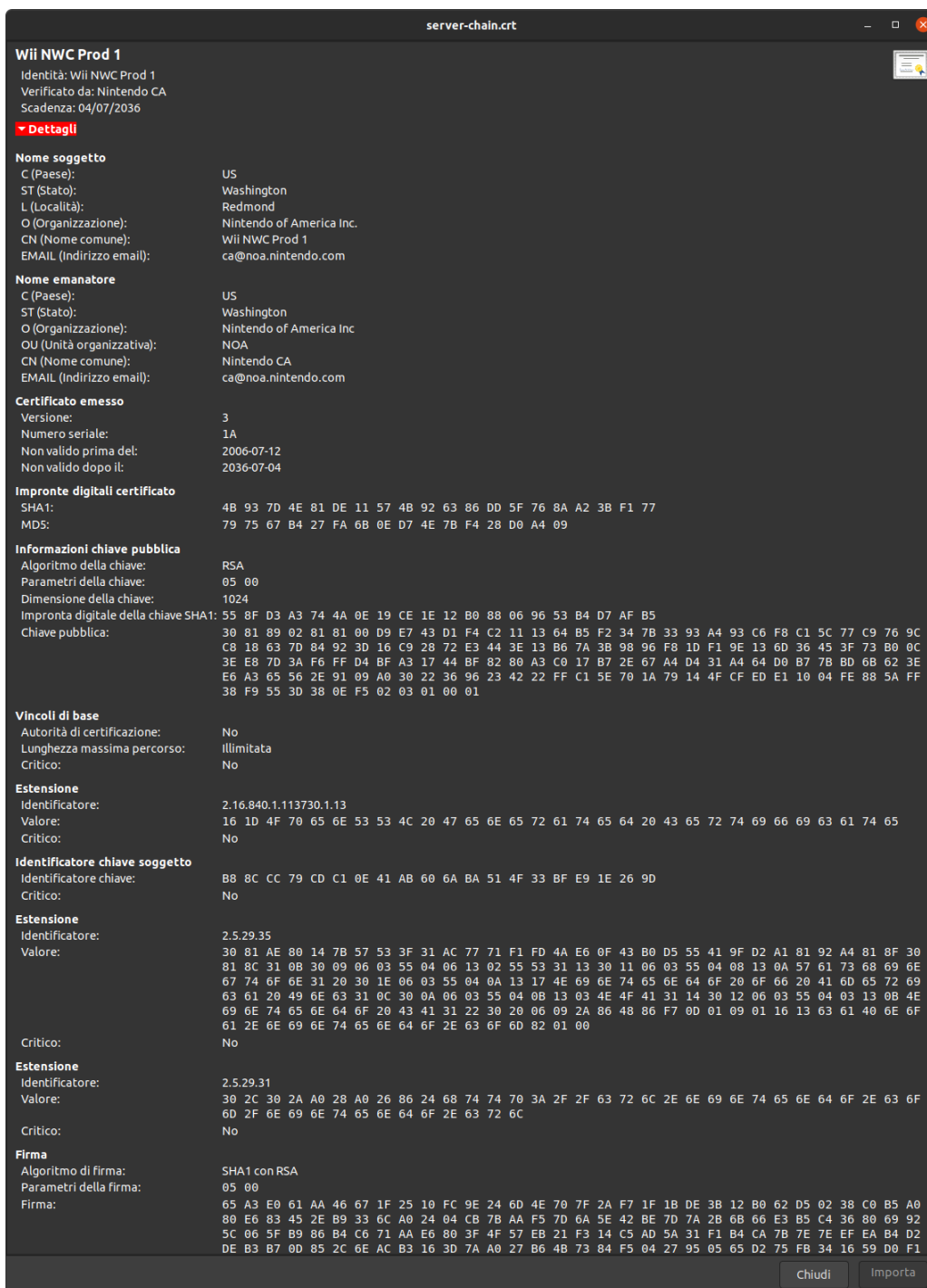


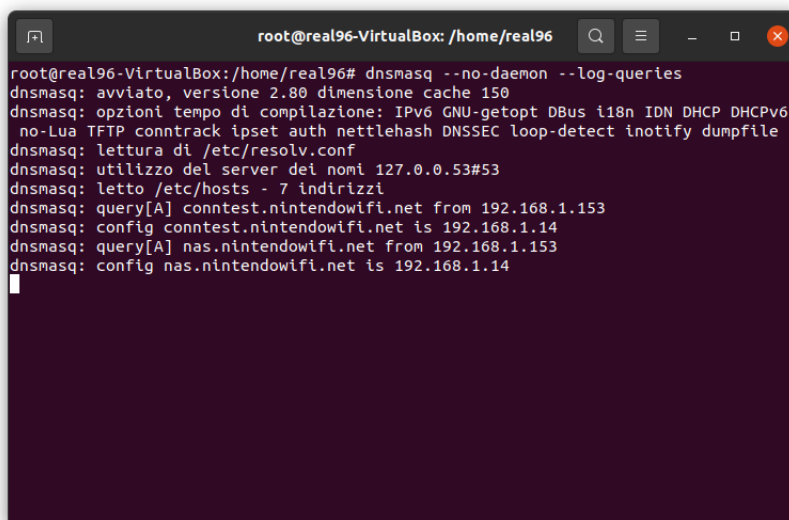
Figura 4.1: Screen del file del certificato estratto da una console Nintendo Wii

5 | Funzionamento dell'attacco

In questa sezione verrà spiegato passo passo il funzionamento dell'attacco e il modo con il quale si riesce a far connettere la console Nintendo DS al finto server Nintendo, facendole credere che questo sia un server ufficiale. A tal fine verrà utilizzato uno bash script il cui codice viene mostrato nella sezione 6.

5.1 Indirizzare la richiesta verso il server finto

Quando si crea una nuova connessione wireless nelle impostazioni Wi-Fi della console, questa esegue prima una prova di connessione al router e successivamente fa una richiesta di tipo GET al server ufficiale Nintendo, verso l'indirizzo *conntest.nintendowifi.net* e verso l'indirizzo *nas.nintendowifi.net*, per verificare se i server Nintendo sono attivi. In Figura 5.1 sono mostrate le richieste fatte dalla console, intercettate tramite *dnsmasq*.



```
root@real96-VirtualBox: /home/real96
root@real96-VirtualBox: /home/real96# dnsmasq --no-daemon --log-queries
dnsmasq: avviato, versione 2.80 dimensione cache 150
dnsmasq: opzioni tempo di compilazione: IPv6 GNU-getopt DBus i18n IDN DHCP DHCPv6
no-Lua TFTP conntrack ipset auth nettlehash DNSSEC loop-detect inotify dumpfile
dnsmasq: lettura di /etc/resolv.conf
dnsmasq: utilizzo del server dei nomi 127.0.0.53#53
dnsmasq: letto /etc/hosts - 7 indirizzi
dnsmasq: query[A] conntest.nintendowifi.net from 192.168.1.153
dnsmasq: config conntest.nintendowifi.net is 192.168.1.14
dnsmasq: query[A] nas.nintendowifi.net from 192.168.1.153
dnsmasq: config nas.nintendowifi.net is 192.168.1.14
```

Figura 5.1: Screen della richiesta della console

Dato che la Nintendo ha cessato qualsiasi tipo di servizio online per la console, al termine del tentativo di connessione, la console mostrerà nello schermo inferiore il

codice errore 20110, comunicando tramite un messaggio che il servizio *Nintendo Wi-Fi Connection* è stato interrotto.

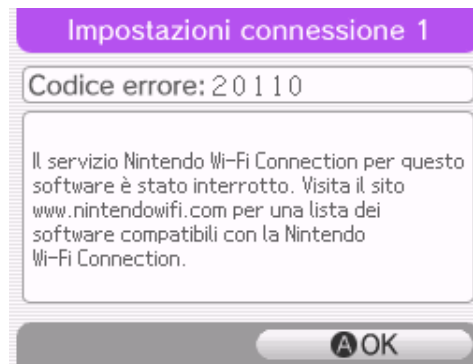


Figura 5.2: Screen dell'errore nella console

Bisogna quindi forzare la console a mandare la richiesta verso il finto server, evitando che questa venga indirizzata verso il server originale Nintendo. Per far ciò, si impostano i DNS statici nella configurazione della connessione della console, disattivando l'opzione *Otteni DNS automaticamente* e inserendo in entrambi gli indirizzi dei server DNS, il primario e il secondario, l'IP della macchina su cui è in esecuzione l'emulatore del server Nintendo. In questo modo, quando la console cercherà di contattare il server Nintendo ufficiale, chiederà all'IP inserito, cioè quello della macchina del finto server Nintendo, di risolvere l'indirizzo.

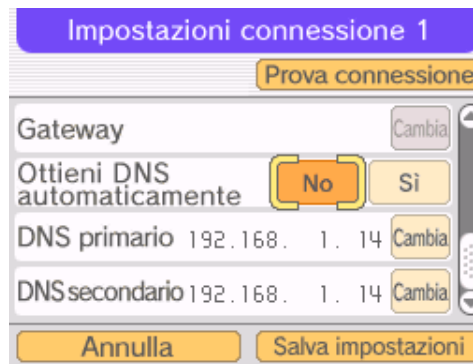


Figura 5.3: Screen della configurazione DNS della console

5.2 Gestione della richiesta

La macchina che ospita il server, però, non è un server DNS e quindi non può automaticamente gestire la richiesta. Per far in modo che ciò accada, è necessario che il

server si comporti anche da server DNS. Questo può essere fatto tramite l'applicazione *dnsmasq*, che permette di creare un server DHCP o DNS nella macchina su cui viene eseguita. Se impostato correttamente, nel momento in cui la console manderà la richiesta di risoluzione dei domini *conntest.nintendowifi.net* e *nas.nintendowifi.net* al server, questo li risolverà puntando verso se stesso.

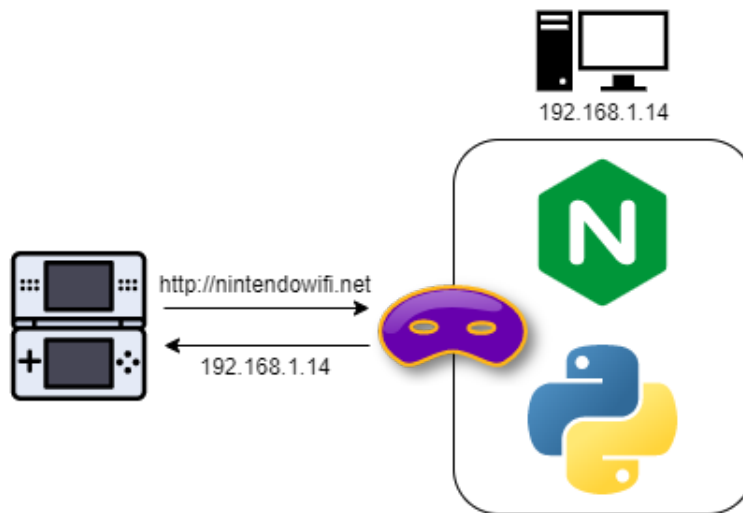


Figura 5.4: Gestione della richiesta DNS da parte di *dnsmasq*

La richiesta verrà poi gestita da *nginx*, web server open source che può essere usato anche come *proxy* inverso. Questo manderà alla console un certificato falso e reindirizzerà la richiesta al server scritto in python tramite la porta specificata nelle impostazioni del virtual host.

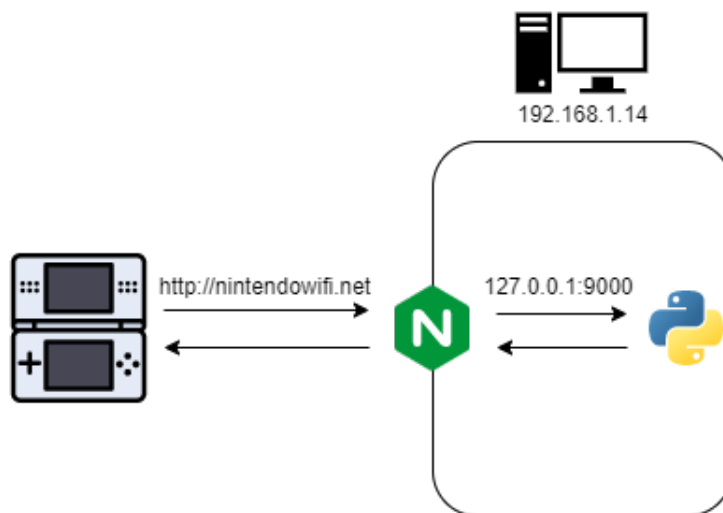


Figura 5.5: Gestione della richiesta HTTP da parte di *nginx*

Il server in python manderà alla console il codice 200, per indicarle che il servizio è ancora attivo e verrà quindi stampato il messaggio di connessione riuscita nello schermo inferiore della console.

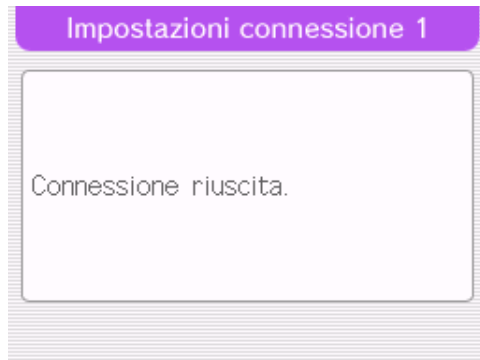


Figura 5.6: Screen della riuscita connessione della console

5.3 Configurazione di nginx

```

1 server {
2     listen 80;
3     listen [::]:80;
4     server_name conntest.nintendowifi.net;
5     location / {
6         proxy_set_header X-Forwarded-Host $host:$server_port;
7         proxy_set_header X-Forwarded-Server $host;

```

```
8     proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
9     proxy_pass http://127.0.0.1:9000;
10 }
```

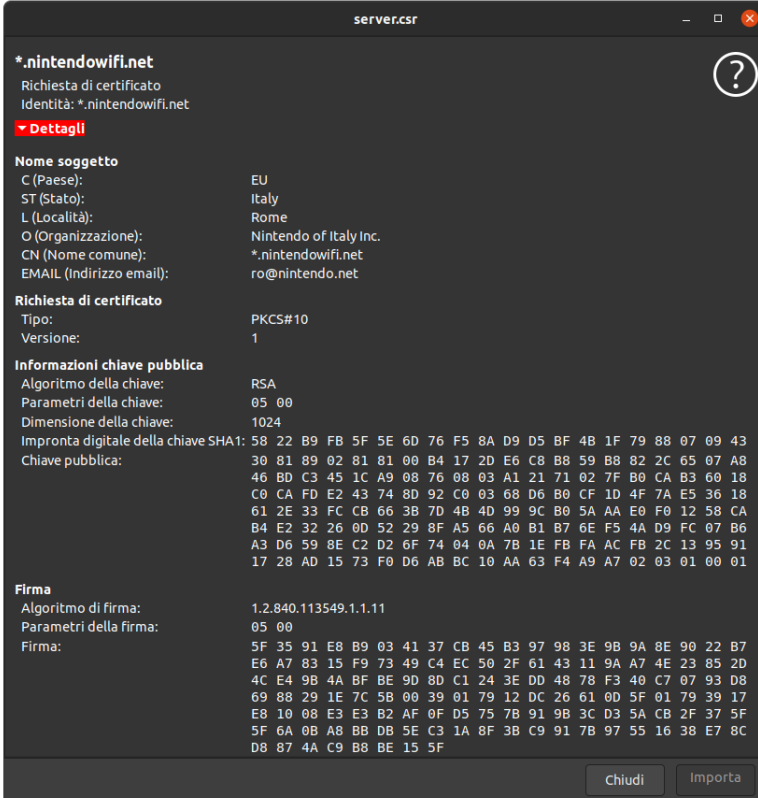
```
1 server {
2     listen 443 ssl;
3     listen [::]:443 ssl;
4     ssl_protocols SSLv3;
5     ssl_ciphers RC4-SHA:RC4-MD5;
6     ssl_certificate /var/www/ssl/server-chain.crt;
7     ssl_certificate_key /var/www/ssl/server.key;
8     server_name naswii.nintendowifi.net nas.nintendowifi.net;
9     underscores_in_headers on;
10    proxy_pass_request_headers on;
11    location / {
12        proxy_set_header X-Forwarded-Host $host:$server_port;
13        proxy_set_header X-Forwarded-Server $host;
14        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
15        proxy_pass http://127.0.0.1:9000;
16    }
17 }
```

Nginx può gestire contemporaneamente più domini, all'interno dello stesso server e indirizzo IP, attraverso i *Server Blocks*. Questi reindirizzano le richieste fatte ad un determinato dominio verso un server, di cui viene specificato il dominio e la porta. Sopra sono mostrate le configurazioni relative alle richieste verso il dominio HTTP *conntest.nintendowifi.net* e i domini HTTPS *naswii.nintendowifi.net* e *nas.nintendowifi.net*. Vengono utilizzate le seguenti opzioni: *listen* per specificare la porta su cui il server deve mettersi in ascolto per le richieste, *server_name* per specificare il dominio verso cui sono indirizzate le richieste che devono essere gestite, *location* per redirigere il traffico della richiesta verso l'indirizzo IP e la porta specificati nell'opzione *proxy_pass*. Nella configurazione relativa alle richieste verso i domini HTTPS viene inoltre specificato, nell'opzione *listen*, il protocollo di sicurezza da usare; vengono utilizzate le opzioni *ssl_protocols* e *ssl_ciphers* per specificare la versione del protocollo di sicurezza e i cifrari da usare per la cifratura e le opzioni *ssl_certificate* e *ssl_certificate_key* per specificare la cartella del certificato che il server *nginx* manderà a qualsiasi client si connetta ad esso, e quella della sua chiave privata.

Quando la console manderà la richiesta verso il dominio HTTP *conntest.nintendowifi.net*, il server *nginx*, che è in ascolto sulla porta 80, cioè quella di HTTP, la reindirizzerà verso l'indirizzo IP 127.0.0.1, che indica l'indirizzo IP della macchina stessa su cui esso è eseguito, e verso la porta 9000, che è associata al processo dell'emulatore del server Nintendo, il quale gestirà e risponderà alla richiesta. La stessa cosa avverrà quando la console manderà la richiesta verso il dominio HTTPS *nas.nintendowifi.net* ma, in questo caso, il server *nginx*, che è in ascolto anche sulla porta 443, ovvero quella di HTTPS, prima di reindirizzare la richiesta verso l'emulatore del server, manderà il certificato falso alla console, in modo che questa possa accettare la connessione.

5.4 Falsificazione del certificato

Per fare in modo che la console si connetta al finto server Nintendo, bisogna generare un certificato che questa possa considerare valido dopo averlo ricevuto e verificato. A tal fine, secondo quanto già spiegato nella sezione 3.6, bisogna generare una coppia di chiavi e una CSR tramite lo script introdotto ad inizio capitolo. In Figura 5.7 viene mostrato il file CSR generato dallo script, contenente informazioni false riguardanti una sede fittizia di Nintendo situata a Roma, al fine di creare il certificato da mandare alla console:



```
server.csr
*.nintendowifi.net
Richiesta di certificato
Identità: *.nintendowifi.net
▼ Dettagli
Nome soggetto
C (Paese): EU
ST (Stato): Italy
L (Località): Rome
O (Organizzazione): Nintendo of Italy Inc.
CN (Nome comune): *.nintendowifi.net
EMAIL (Indirizzo email): ro@nintendo.net
Richiesta di certificato
Tipo: PKCS#10
Versione: 1
Informazioni chiave pubblica
Algoritmo della chiave: RSA
Parametri della chiave: 05 00
Dimensione della chiave: 1024
Impronta digitale della chiave SHA1: 58 22 B9 FB 5F 5E 6D 76 F5 8A D9 D5 BF 4B 1F 79 88 07 09 43
30 81 89 02 81 81 00 B4 17 2D E6 C8 B8 59 B8 82 2C 65 07 A8
46 BD C3 45 1C A9 08 76 08 03 A1 21 71 02 7F B0 CA B3 60 18
C0 CA FD E2 43 74 8D 92 C0 03 68 D6 B0 CF 1D 4F 7A E5 36 18
61 2E 33 FC CB 66 3B 7D 4B 4D 99 9C B0 5A AA E0 F0 12 58 CA
B4 E2 32 26 0D 52 29 8F A5 66 A0 B1 B7 6E F5 4A D9 FC 07 B6
A3 D6 59 0E C2 D2 6F 74 04 0A 7B 1E FB FA AC FB 2C 13 95 91
17 28 AD 15 73 F0 D6 AB BC 10 AA 63 F4 A9 A7 02 03 01 00 01
Chiave pubblica:
Firma
Algoritmo di firma: 1.2.840.113549.1.1.11
Parametri della firma: 05 00
Firma:
5F 35 91 E8 B9 03 41 37 CB 45 B3 97 98 3E 9B 9A 8E 90 22 B7
E6 A7 83 15 F9 73 49 C4 EC 50 2F 61 43 11 9A A7 4E 23 85 2D
4C E4 9B 4A BF BE 9D 8D C1 24 3E DD 48 78 F3 40 C7 07 93 D8
69 88 29 1E 7C 5B 00 39 01 79 12 DC 26 61 0D 5F 01 79 39 17
E8 10 08 E3 E3 B2 AF 0F D5 75 7B 91 9B 3C D3 5A CB 2F 37 5F
5F 6A 0B A8 BB DB 5E C3 1A 8F 3B C9 91 7B 97 55 16 38 E7 8C
D8 87 4A C9 B8 BE 15 5F
Chiudi Importa
```

Figura 5.7: Screen del file della CSR

Successivamente, partendo dal file CSR mostrato nell'immagine, si procede con la creazione del certificato. Come spiegato nella sezione 4.1, per generare e firmare un certificato che la console Nintendo DS possa considerare valido, bisogna utilizzare un certificato firmato dalla CA di Nintendo con la relativa chiave privata. Questo non deve necessariamente essere un certificato di una CA intermedia, può essere un qualsiasi certificato valido, firmato dalla CA di Nintendo. Un certificato con queste caratteristiche è quello della console Nintendo Wii, la cui estrazione, con la relativa chiave privata, è spiegata nella sezione 7.3. Questo certificato presenta con valore

False la variabile booleana *Vincoli di base* descritta nella sezione 3.2, quindi non è un certificato che può essere usato per firmarne altri. Ma dato che nell'implementazione di SSL fatta da Nintendo la variabile non viene controllata, i certificati firmati risulteranno validi.

Una volta ottenuto il certificato firmato da Nintendo con la relativa chiave privata, si procede con la generazione e la firma di un altro certificato da mandare alla console Nintendo DS, la quale potrà effettuare la verifica, riuscendo a risalire tutta la catena di certificazione, fino al certificato di root. Il processo, con gli specifici comandi usati, è spiegato nella sezione 6.7.

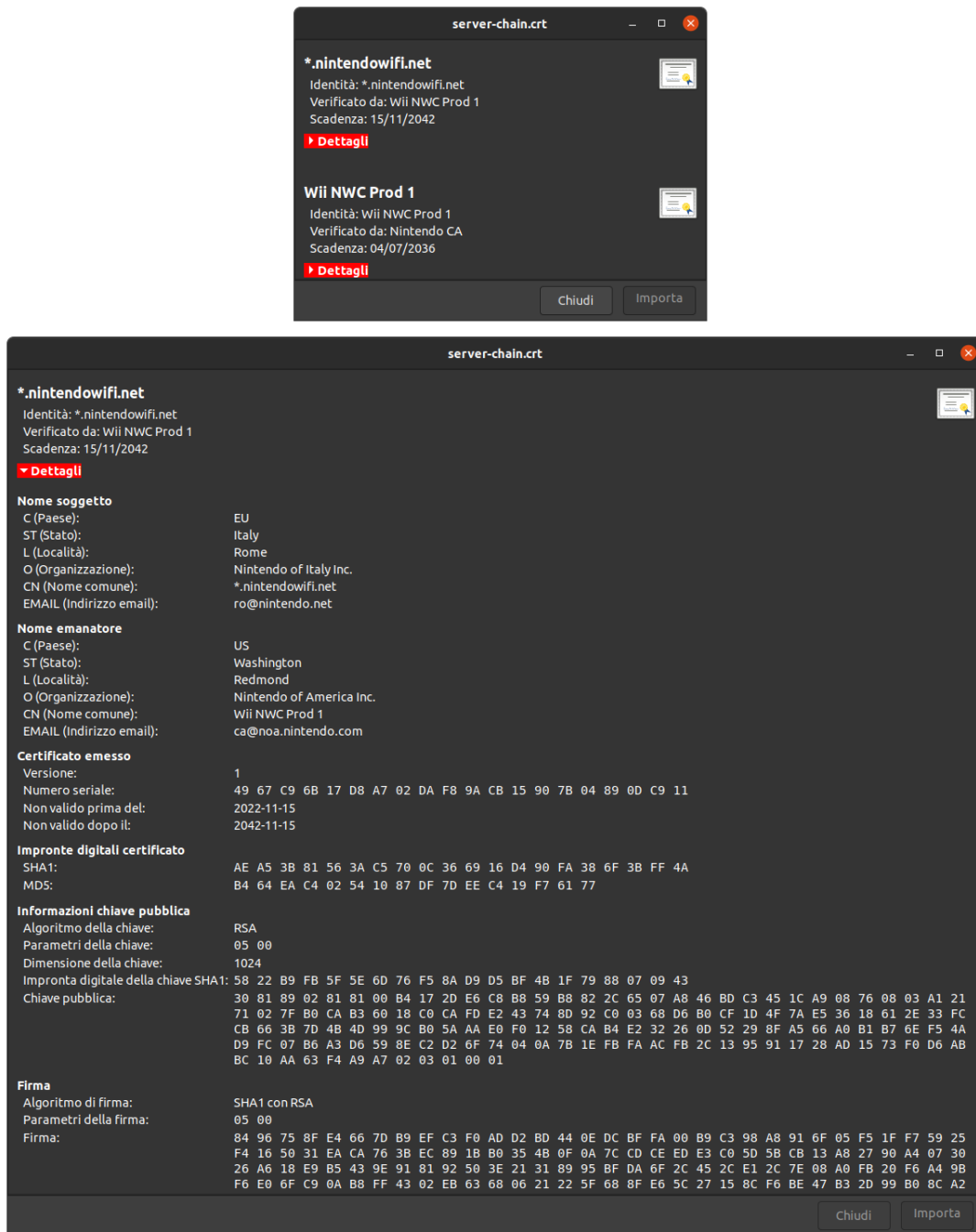


Figura 5.8: Screen del file del certificato mandato alla console

6 | Descrizione dello script

In questa sezione verrà mostrato e spiegato il codice che compone lo script.

6.1 checkOSVersion

```
1 function checkOSVersion() {
2   if [[ "$NAME" == "Ubuntu" && $VERSION_ID < 14 ]] || [[ "$NAME" == "Debian GNU/Linux" && $VERSION_ID < 9 ]] ||
3     [[ "$NAME" != "Ubuntu" && "$NAME" != "Debian GNU/Linux" ]];
4   then
5     echo -e "This Linux distro is not currently supported!\nSupported distro: Ubuntu 14+ or Debian 9+"
6
7     exit 1
8   fi
9 }
```

Tramite questa funzione, lo script controlla se la versione del Sistema Operativo, sul quale esso viene eseguito, è compatibile o meno. Se la versione non è compatibile, lo script stampa sul terminale le versioni dei Sistemi Operativi supportate e successivamente termina la sua esecuzione.

6.2 checkCertFile

```
1 function checkCertFile() {
2   if [ ! -f "00000011.app" ]; then
3     echo -e "Unable to find \"00000011.app\" NAND file!\nBe sure to put it in the same path of the script\n"
4
5     exit 1
6   fi
7 }
```

Tramite questa funzione, lo script controlla se è presente il file *00000011.app* nella stessa cartella in cui esso si trova. Nel caso il file non fosse presente, lo script non va avanti e stampa sul terminale di non averlo trovato. Il comando **-f**, seguito da un parametro che indica il nome del file, serve a verificare se tale file esiste o meno. Il file è fondamentale per l'installazione, in quanto contiene al suo interno il certificato della Wii firmato da Nintendo e la chiave privata della console, i quali verranno estratti e utilizzati per generare il certificato falso. Nella sezione 7.3 viene spiegato come poterli ottenere.

6.3 checkSudo

```
1 function checkSudo() {
2   if [ "$(id -u)" != "0" ]; then
3     exec sudo "$0"
4   fi
5 }
```

Molti dei comandi che lo script dovrà eseguire necessitano dei permessi di root. Tramite la funzione suindicata, lo script controlla se è stato eseguito dall'utente con tali permessi e, in caso contrario, imposta i privilegi di root, chiedendo all'utente di inserire la propria password utente. Il comando **id** stampa le informazioni dell'utente indicato come parametro oppure, se nessun nome utente è specificato come argomento, dell'utente loggato in quel momento. Tramite il parametro **-u** viene stampato l'ID utente che, nel caso di utente con privilegi di root, equivale a 0. Quindi, tramite l'**if**, viene confrontato con 0 il valore di **id -u**. Si controlla così se lo script è stato eseguito con i privilegi di root e, in caso contrario, si esegue il comando **exec sudo** passando **\$0** come parametro, il che equivale al nome dello script stesso.

6.4 checkSources

```
1 function checkSources() {
2   echo "Checking if all sources are reachable..."
3
4   if ping -c 2 github.com >/dev/nul && ping -c 2 www.openssl.org >/dev/nul &&
5     ping -c 2 www.nginx.com >/dev/nul && ping -c 2 bootstrap.pypa.io >/dev/nul;
6   then
7     echo -e "All needed sources are reachable!\n"
8   else
9     echo -e "Unable to reach all the needed sources!\n"
10
11     exit 1
12   fi
13 }
```

Tramite questa funzione, lo script controlla se i siti da cui deve scaricare i vari codici risorsa sono raggiungibili. Il comando **ping**, seguito da un indirizzo IP, serve a determinare se tale IP, situato in una rete locale o pubblica, può essere raggiunto. Per determinare la raggiungibilità di un IP, vengono inviati quattro pacchetti ICMP, con una richiesta di tipo echo di 32 byte ciascuno, all'indirizzo indicato come parametro. Se il server è attivo, risponderà mandando dei pacchetti a sua volta. Il parametro **-c** serve a specificare dopo quanti pacchetti ricevuti dal server il ping deve terminare. In questo caso, ci si accontenta di due pacchetti da parte di ogni server. Al contrario, se uno dei siti necessari non è raggiungibile, lo script termina la sua esecuzione.

6.5 getPackages

```

1 function getPip() {
2   wget https://bootstrap.pypa.io/pip/2.7/get-pip.py
3   chmod 777 get-pip.py
4   python2.7 get-pip.py
5   rm get-pip.py
6 }

```

```

1 function getPackages() {
2   echo "Installing required packages..."
3   dpkg --configure -a
4   apt-get update
5   apt-get install make gcc python2.7 dnsmasq git net-tools wget -y
6
7   if [ "$version" == "Ubuntu 20.04" ] || [ "$version" == "Ubuntu 22.04" ] ||
8     [ "$version" == "Debian GNU/Linux 11" ];
9   then
10    getPip
11    pip install twisted
12  else
13    apt-get install python-twisted -y
14  fi
15
16  echo
17 }

```

Tramite questa funzione, lo script scarica e installa tutti i pacchetti necessari per il funzionamento del server. Tramite il comando **apt-get install**, vengono scaricati i seguenti tool: **make** e **gcc** utilizzati per compilare **openssl** ed **nginx**, **python2.7** utilizzato per far partire l'emulatore del server Nintendo, **dnsmasq** per creare il server DNS, **git** per scaricare da Github il progetto dell'emulatore del server Nintendo e il file di configurazione di *nginx*, **wget** per scaricare file. Per poter scaricare **python-twisted**, necessario per il funzionamento dell'emulatore del server, viene fatto un controllo della versione di Linux. Nel caso di versione superiore ad Ubuntu 18.04 o a Debian 10, sarà necessario installare e utilizzare **pip** per poter installare questo framework.

6.6 extractCertificateFiles

```

1 function extractCertificateFiles() {
2   echo "Extracting certificate files from \"00000011.app\"..."
3   mkdir -p /var/www/ssl/
4   cp 00000011.app /var/www/ssl/
5   cd /var/www/ssl/
6   wget https://github.com/Real96/Wii_extract_certs/releases/download/linux/extract_certs
7   chmod 777 extract_certs
8   ./extract_certs 00000011.app
9   rm !("clientca.pem"|"clientcakey.pem")
10  echo
11 }

```

Tramite questa funzione, lo script scarica da Github l'eseguibile **extract_certs**, per mezzo del quale estrae il certificato Nintendo, con la relativa chiave, dal file *00000011.app* e li posiziona all'interno della cartella */var/www/ssl*. Il comando **mkdir -p** crea una cartella nel percorso specificato come parametro. Se le cartelle che compongono il percorso specificato non esistono, vengono create. Il comando **cp** copia il file, che è specificato nel primo parametro, all'interno del percorso specificato come secondo parametro. Tramite il comando **wget** viene scaricata da Github l'applicazione *extract_certs*, alla quale vengono aggiunti i permessi di lettura, scrittura ed esecuzione tramite il comando **chmod**. Quest'applicazione può essere eseguita tramite il comando **./extract_certs 00000011.app**, per mezzo del quale estrae dal file specificato come parametro il certificato della Wii con la relativa chiave privata e il certificato di root Nintendo. Attraverso il comando **rm !("clientca.pem"|"clientcakey.pem")**, vengono eliminati tutti i file presenti all'interno della cartella, tranne **clientca.pem**, che rappresenta il certificato della console Wii, e **clientcakey.pem**, che ne rappresenta la relativa chiave privata.

Il file *clientca.pem* e il file *clientcakey.pem* sono rispettivamente i contenitori del certificato della Nintendo Wii e della relativa chiave privata, entrambi in formato *der*.

6.7 forgeCertificateFiles

```

1 function forgeCertificateFiles() {
2   echo "Forging fake certificate files..."
3   openssl rsa -inform der -in clientcakey.pem -out NWC.key
4   openssl x509 -inform der -in clientca.pem -out NWC.crt
5   openssl genrsa -out server.key 1024
6   openssl req -new -key server.key -out server.csr <<EOF
7   EU
8   Italy
9   Rome
10  Nintendo of Italy Inc.
11  .
12  *.nintendowifi.net
13  ro@nintendo.net
14  .
15  .
16  EOF
17  openssl x509 -req -in server.csr -CA NWC.crt -CAkey NWC.key -CAcreateserial -out server.crt -days 7305 -sha1
18  cat server.crt NWC.crt > server-chain.crt
19  rm !("server.key"|"server-chain.crt")
20  echo
21 }
```

Tramite questa funzione, lo script genera il certificato falso che verrà mandato dal server *nginx* alla console Nintendo DS, una volta che questa avrà inviato una richiesta di connessione. A tal fine viene utilizzato *openssl*, un kit di strumenti che implementa tutte le versioni del protocollo TLS. Tramite il comando **openssl rsa -inform der -in clientcakey.pem -out NWC.key** viene preso in input il file *clientcakey.pem*, estratto dalla funzione precedente, viene convertito in un formato compatibile con il server *nginx* e viene salvato nel file *NWC.key*. Attraverso il comando **openssl x509 -inform der -in clientca.pem -out NWC.crt** viene preso in input il file *clientca.pem*, estratto dalla funzione precedente, viene convertito in un formato compatibile con il server

nginx e viene salvato nel file *NWC.crt*. Tramite i comandi **openssl genrsa -out server.key 1024** e **openssl req -new -key server.key -out server.csr** viene eseguito in pratica quanto spiegato nella sezione 5.4, cioè vengono generate una coppia di chiavi da 1024 bit, una privata e una pubblica, all'interno del file *server.key* e viene creata una CSR all'interno del file *server.csr*, inserendo informazioni false riguardanti una sede fittizia di Nintendo situata a Roma. Attraverso il comando **openssl x509 -req -in server.csr -CA NWC.crt -CAkey NWC.key -CAcreateserial -out server.crt -days 7305 -sha1** viene generato e firmato il certificato falso. A tal fine vengono usati i seguenti parametri: **x509**, utility multifunzione riguardante i certificati, **-req** per passare in input il file della richiesta di firma, **-CA** e **-CAkey** per passare in input i file generati dal comando precedente, ovvero il certificato della Nintendo Wii e la relativa chiave privata, che verrà utilizzata per firmare, **-CAcreateserial** per creare un file contenente un numero seriale, generato randomicamente, che verrà utilizzato per la creazione del certificato, **-out** per indicare in quale file salvare il certificato una volta generato e firmato, **-days** per indicare la durata del certificato, **-sha1**, per indicare il nome del tipo di *digest* da usare. Tramite il comando **cat server.crt NWC.crt > server-chain.crt** vengono uniti, all'interno del file *server-chain.crt*, il certificato appena creato con il certificato della Nintendo Wii, operazione necessaria per fare in modo che il certificato venga accettato dalla console. Infine, tramite il comando **rm !("server.key"|"server-chain.crt")**, vengono rimossi tutti i file presenti nella cartella, tranne *server.key*, utilizzato come chiave privata del server *nginx*, e *server-chain.crt*, cioè il certificato che verrà mandato alla console Nintendo DS.

6.8 buildOpensslNginx

```

1 function setOpensslConf() {
2     sed -i '1iopenssl_conf = default_conf' /usr/local/ssl/openssl.cnf
3     cat >> /usr/local/ssl/openssl.cnf <<EOF
4
5     [default_conf]
6     ssl_conf = ssl_sect
7
8     [ssl_sect]
9     system_default = system_default_sect
10
11    [system_default_sect]
12    CipherString = ALL:@SECLEVEL=0
13    EOF
14    ldconfig
15 }

```

```

1 function buildOpenssl() {
2     cd /var/www/openssl-1.1.1s/
3     ./config enable-ssl3 enable-ssl3-method enable-weak-ssl-ciphers
4     make
5     make install
6     setOpensslConf
7 }

```

```

1 function buildNginx() {
2   cd /var/www/nginx-1.23.2/
3   apt-get install libpcre3 libpcre3-dev zlib1g zlib1g-dev -y
4   ./configure --with-http_ssl_module --with-ld-opt="-L/usr/local"
5   make
6   make install
7 }

```

```

1 function buildOpenSSLNginx() {
2   cd /var/www/
3
4   if [ "$version" == "Ubuntu 14.04" ]; then
5     echo "Installing nginx..."
6     apt-get install nginx -y
7   else
8     echo "Building openssl and nginx enabling weak ciphers..."
9     wget http://nginx.org/download/nginx-1.23.2.tar.gz https://www.openssl.org/source/openssl-1.1.1s.tar.gz
10    cat *.tar.gz | tar -xzf -
11    chmod -R 777 .
12    rm *.tar.gz
13    buildOpenSSL
14    buildNginx
15    cd /var/www/
16    rm -r openssl-1.1.1s nginx-1.23.2
17  fi
18
19  echo
20 }

```

Tramite questa funzione, lo script, a seconda della versione di Linux, installa *nginx* e *openssl*, oppure scarica e compila i loro codici sorgente, abilitando i vecchi cifrari e creando un link tra loro che permette a *nginx* di usare la versione appena compilata di *openssl*. Dalla versione di Ubuntu 16.04 e Debian 9 in poi, in *openssl* sono stati disabilitati il protocollo SSLv3 e i cifrari *RC4-SHA* e *RC4-MD5*, necessari per la comunicazione con la console. Quindi, per poterli abilitare nuovamente nelle versioni più recenti di Linux, è necessario scaricare e compilare il codice risorsa di *openssl* con le impostazioni spiegate successivamente. Inoltre occorre compilare il codice risorsa di *nginx*, indicandogli di usare la versione di *openssl* appena compilata, dato che, se installato normalmente, *nginx* utilizzerà la versione di *openssl* preinstallata nel sistema. A tal fine, verrà fatto un controllo iniziale della versione di Linux e, in caso si tratti di Ubuntu 14.04, lo script installerà normalmente *nginx*, tramite il comando **apt-get install nginx -y**, e l'emulatore del server utilizzerà la versione di *openssl* preinstallata nel sistema. Nel caso non si tratti di Ubuntu 14.04, tramite i comandi **wget http://nginx.org/download/nginx-1.23.2.tar.gz https://www.openssl.org/source/openssl-1.1.1s.tar.gz** e **cat *.tar.gz | tar -xzf -** verranno scaricati e scompattati i codici risorsa di *nginx* e *openssl*. Successivamente verranno eseguite le funzioni **buildOpenSSL** e **buildNginx**, che compileranno e configureranno i due software.

La funzione **buildOpenSSL** entra nella cartella del codice risorsa di *openssl* attraverso il comando **cd /var/www/openssl-1.1.1s/**, abilita l'utilizzo di SSLv3 e dei vecchi cifrari tramite il comando **./config enable-ssl3 enable-ssl3-method enable-weak-ssl-ciphers** e installa *openssl* usando i comandi **make** e **make install**. Il software viene poi configurato per mezzo della funzione **setOpenSSLConf**, che abilita l'utilizzo delle vecchie versioni dei protocolli di sicurezza sopra citati, modificando il file di

configurazione `/usr/local/ssl/openssl.cnf` e aggiungendo l'opzione `CipherString = ALL:@SECLEVEL=0`, la quale abbassa al minimo il livello di sicurezza del software. Infine, la funzione `setOpenesslConf` carica il file di configurazione appena citato, utilizzando il comando `ldconfig`.

La funzione `buildNginx` entra nella cartella del codice risorsa di `nginx` attraverso il comando `cd /var/www/nginx-1.23.2/`, installa le librerie necessarie per la compilazione di tale codice tramite il comando `apt-get install libpcre3 libpcre3-dev zlib1g zlib1g-dev -y`, abilita l'utilizzo di SSL e crea un link con la corretta versione di `openssl` usando il comando `./configure --with-http_ssl_module --with-ld-opt="-L/usr/local"` e infine compila e installa il software per mezzo dei comandi `make` e `make install`.

6.9 createNginxNintendoSb

```

1 function createNginxNintendoSb() {
2   echo "Creating Nintendo server blocks..."
3   git clone https://github.com/Real96/Nintendo-DWC-Installer-Script
4
5   if [ "$version" == "Ubuntu 14.04" ]; then
6     mv /var/www/Nintendo-DWC-Installer-Script/nginx_conf/14/dwc-hosts /etc/nginx/sites-available/
7     echo -e "Done!\nenabling...\n"
8     ln -s /etc/nginx/sites-available/dwc-hosts /etc/nginx/sites-enabled
9     service nginx restart
10  else
11    mv -f /var/www/Nintendo-DWC-Installer-Script/nginx_conf/16+/nginx.conf /usr/local/nginx/conf/
12    echo -e "Done!\nenabling...\n"
13    /usr/local/nginx/sbin/nginx
14  fi
15 }
```

Tramite questa funzione, lo script configura il server `nginx` secondo quanto spiegato nella sezione 5.3. Tramite il comando `git clone` viene scaricato il progetto dello script, in cui si trovano i file di configurazione di cinque `server blocks`. Questi, a seconda della versione di Linux, vengono spostati tramite il comando `mv` all'interno della cartella in cui si trovano i file di configurazione di `nginx`. Nel caso di Ubuntu 14.04, occorre abilitare la configurazione, eseguendo il comando `ln -s /etc/nginx/sites-available/dwc-hosts /etc/nginx/sites-enabled`. Viene infine avviato il server `nginx` tramite i `service nginx restart` nel caso di Ubuntu 14.04, oppure `/usr/local/nginx/sbin/nginx` in tutte le altre versioni di Linux.

6.10 getDWCRRepo

```

1 function getDWCRRepo() {
2   echo "Downloading the DWC server..."
3   git clone https://github.com/Real96/dwc_network_server_emulator
4   chmod -R 777 .
5   echo -e "Done!\nenabling..."
6   cd /var/www/dwc_network_server_emulator/
7   (python2.7 master_server.py &) &>/dev/null
```

```

8  sleep 3
9  clear
10 }
```

Tramite questa funzione, lo script scarica l'emulatore del server Nintendo e lo avvia. L'emulatore viene scaricato, tramite il comando **git clone**, dal link del progetto presente su Github, all'interno della cartella `/var/www`. Attraverso il comando **chmod -R 777** . vengono modificati i permessi di tutti i file presenti nella cartella e nelle sottocartelle, settando i permessi di lettura, scrittura ed esecuzione. Tramite il comando **cd** ci si sposta dentro la cartella dell'emulatore del server Nintendo. Infine, attraverso comando (**python2.7 master_server.py &**) **&>/dev/null** si avvia il server utilizzando *python2.7*. Il parametro **&** permette l'esecuzione in background del processo dell'emulatore, mentre il parametro **&>/dev/null** evita che l'output venga stampato sulla console del terminale, riversandolo tutto verso il file `/dev/null`, il quale cancella in automatico qualsiasi cosa venga scritta su di esso.

6.11 configDnsmasq

```

1 function configDnsmasq() {
2   echo -e "-----dnsmasq configuration-----\nYour LAN IP is:"
3   hostname -I
4   echo "Your public IP is:"
5   wget -q -O - ipinfo.io/ip
6   echo -e "\nType in your IP:"
7   read -re IP
8   cat >>/etc/dnsmasq.conf <<EOF
9
10 EOF
11
12 if [ "$version" == "Ubuntu 18.04" ] || [ "$version" == "Ubuntu 20.04" ] ||
13    [ "$version" == "Ubuntu 22.04" ];
14 then
15   cat >>/etc/dnsmasq.conf <<EOF
16 bind-dynamic
17 EOF
18 fi
19
20   cat >> /etc/dnsmasq.conf <<EOF
21 address=/nintendowifi.net/$IP
22 listen-address=$IP,127.0.0.1
23 EOF
24   echo -e "\ndnsmasq setup completed!\nenabling...\n"
25   service dnsmasq restart >/dev/nul
26 }
```

Per mezzo di questa funzione, lo script configura e riavvia *dnsmasq*. Tramite il comando **hostname -I** viene stampato l'indirizzo IP locale della macchina. Utilizzando il comando **wget -q -O - ipinfo.io/ip** viene fatta una richiesta al dominio `https://ipinfo.io/ip` per ottenere l'indirizzo pubblico della macchina che viene poi stampato sulla console del terminale. Tramite il comando **echo** viene stampata la richiesta

di inserimento dell'indirizzo IP, che può essere sia locale che pubblico, a seconda dell'utilizzo che si vuole fare del server. Usando il comando **read -re IP** viene inizializzata la variabile *IP* con l'indirizzo IP inserito. Tramite il comando **cat** vengono aggiunte al file di configurazione di *dnsmasq* le impostazioni necessarie affinché l'applicazione possa gestire correttamente le richieste della console, come spiegato nella sezione 5.2. Pertanto viene fatto un controllo della versione di Linux e, nel caso di versione di Ubuntu superiore alla 16.04, viene aggiunta l'opzione **bind-dynamic**, che rende comunque possibile l'utilizzo della porta 53 del servizio DNS, nonostante il sistema veda questa porta già occupata. Questa opzione permette di associare a *dnsmasq* l'indirizzo delle singole interfacce, consentendo una sua esecuzione su più istanze e, nel caso appaiano nuove interfacce o nuovi indirizzi, l'applicazione si mette automaticamente in ascolto su questi. Vengono poi aggiunte le opzioni **address=/nintendowifi.net/\$IP**, che permette di reindirizzare tutte le richieste fatte al dominio specificato verso l'indirizzo IP specificato, e **listen-address=\$IP,127.0.0.1**, che consente a *dnsmasq* di mettersi in ascolto sugli indirizzi specificati. In entrambe le opzioni **\$IP** è il valore della variabile contenente l'indirizzo IP inserito precedentemente. Infine viene riavviato *dnsmasq* tramite il comando **service dnsmasq restart >/dev/nul**, evitando di stampare l'output sulla console del terminale.

6.12 addCronJob

```

1 function addCronJob() {
2   read -p "Would you like to set a cron job to automatically run the server at system boot? [y/n] " yn
3
4   if [ "$yn" != "${yn#[Yy]}" ];then
5     apt-get install cron -y
6
7     if [ "$version" == "Ubuntu 14.04" ]; then
8       mv /var/www/Nintendo-DWC-Installer-Script/cron_job/14/start_dwc_server.sh /
9     else
10      mv /var/www/Nintendo-DWC-Installer-Script/cron_job/16+/start_dwc_server.sh /
11    fi
12
13    chmod 777 /start_dwc_server.sh
14    echo "@reboot sh /start_dwc_server.sh" >/tmp/dwc-cron
15    crontab -u $USER /tmp/dwc-cron
16    echo -e "\nCron job created!\n"
17  fi
18
19  rm -r /var/www/Nintendo-DWC-Installer-Script
20 }
```

Tramite questa funzione, lo script imposta un *cron job*, cioè un'attività schedulata che permette l'esecuzione automatica dell'emulatore del server Nintendo all'avvio del sistema. Inizialmente viene stampata sulla console la richiesta di creare il *cron job* e, in caso di risposta affermativa, attraverso il comando **apt-get install cron -y**, la funzione procederà con l'installazione di **cron**, utility che permette di schedare l'esecuzione automatica di script o comandi in background. Successivamente viene fatto un controllo della versione di Linux, per stabilire la versione corretta dello script *start_dwc_server.sh*, il quale va eseguito ogni volta che la macchina viene

riavviata. Entrambe le versioni dello script si troveranno all'interno della cartella del progetto scaricato in precedenza dalla funzione mostrata nella sezione 6.9. Questo script contiene al suo interno i comandi per avviare *nginx*, *dnsmasq* e l'emulatore del server Nintendo. Tramite il comando **chmod 777 /start_dwc_server.sh** vengono aggiunti allo script i permessi di lettura, scrittura ed esecuzione, necessari per il suo funzionamento. Attraverso il comando **echo "@reboot sh /start_dwc_server.sh" >/tmp/dwc-cron**, viene scritta, all'interno del file */tmp/dwc-cron*, la regola **@reboot sh /start_dwc_server.sh**, la quale indica al Sistema Operativo di eseguire, al suo avvio, lo script *start_dwc_server*, utilizzando la *shell* di Linux. Infine, tramite il comando **crontab -u \$USER /tmp/dwc-cron**, il *cron job* */tmp/dwc-cron* viene abilitato per l'utente su cui è in esecuzione lo script principale.

6.13 endMessage

```
1 function endMessage() {
2     echo -e "Done! Your personal DWC server is now ready!\n"
3 }
```

Tramite questa funzione, lo script comunica che il processo di installazione è terminato, stampando un messaggio sulla console del terminale.

7 | Esecuzione dell'attacco

L'attacco può essere diviso in due fasi. Nella prima fase bisogna ottenere il certificato foglia di una qualsiasi Nintendo Wii, firmato dalla CA radice di Nintendo, presente all'interno del file *00000011.app*; questo file è estraibile da un dump della NAND della console appena citata. Questo certificato serve per poter creare e firmare un finto certificato Nintendo, che verrà usato dall'emulatore del server Nintendo per poter instaurare la connessione con la console Nintendo DS. Successivamente, in un PC o in una macchina virtuale, bisogna installare una versione di Linux Ubuntu 14 o superiore, oppure una versione di Linux Debian 9 o superiore. L'attacco potrebbe funzionare su qualsiasi versione di Linux ma, allo stato attuale, lo script supporta solo le versioni sopra citate, in quanto le uniche testate. Nel caso si installi il sistema operativo in una macchina virtuale, bisogna assicurarsi di impostare la scheda di rete in modalità *bridge*, in modo tale che il sistema utilizzi direttamente l'hardware del PC che la ospita e non una scheda di rete virtuale.

Nella seconda fase si esegue lo script. Questo installerà tutti i software necessari, tra cui *openssl* e *nginx*, compilati entrambi con l'attivazione dei vecchi standard di sicurezza, gli unici supportati dalla console e che le permettono di instaurare una connessione. Lo script, dopo aver impostato i file di configurazione dei software sopra citati, chiederà all'utente di inserire l'indirizzo IP, locale o pubblico, della macchina che ospiterà il server. Da questa scelta dipenderà il tipo di accesso al server: se l'utente inserisce l'indirizzo IP pubblico, tramite questo ogni console potrà avere accesso al server dall'esterno, e quindi potrà connettersi da qualsiasi parte del mondo; se invece l'utente inserisce l'indirizzo IP locale della macchina, tramite questo ogni console potrà avere accesso al server solo dall'interno della sua stessa sottorete, quindi solo se connessa allo stesso WiFi a cui è connessa la macchina. Lo script, successivamente, scaricherà l'emulatore del server Nintendo dal progetto presente su Github e lo avvierà. Infine, dopo aver avviato il server, chiederà all'utente di creare un *cronjob* all'avvio di sistema, ovvero una routine che avvia in automatico il server ogni qualvolta che la macchina venga accesa.

7.1 Requisiti

Per eseguire lo script che installa l'emulatore del server e per connettersi a quest'ultimo, sono necessari i seguenti requisiti:

- PC o macchina virtuale con Sistema Operativo Linux Ubuntu 14 o superiore oppure Linux Debian 9 o superiore.
- Nintendo Wii con un CFW installato.
- WiFi con protezione WEP o non protetto, con modalità wireless 802.11b e frequenza 2.4 GHz, le uniche supportate dalla console.

7.2 Estrazione della NAND

Per eseguire un dump della NAND della console Wii, è necessario che su tale console sia installato un CFW, in modo tale da avere accesso al menu dell'**Homebrew**¹. Da questo si potrà andare nel menu di *BootMii*, software che permette di eseguire il dump della NAND della console Wii utilizzata, tramite un'apposita opzione. Una volta avviato il processo e terminato il dump, il file della NAND verrà salvato nella scheda SD della console.



Figura 7.1: Screen del menu di BootMii

¹Homebrew: tipo di hack che permette alle console di far girare software che non sono sotto licenza o concessi dalla console Nintendo.

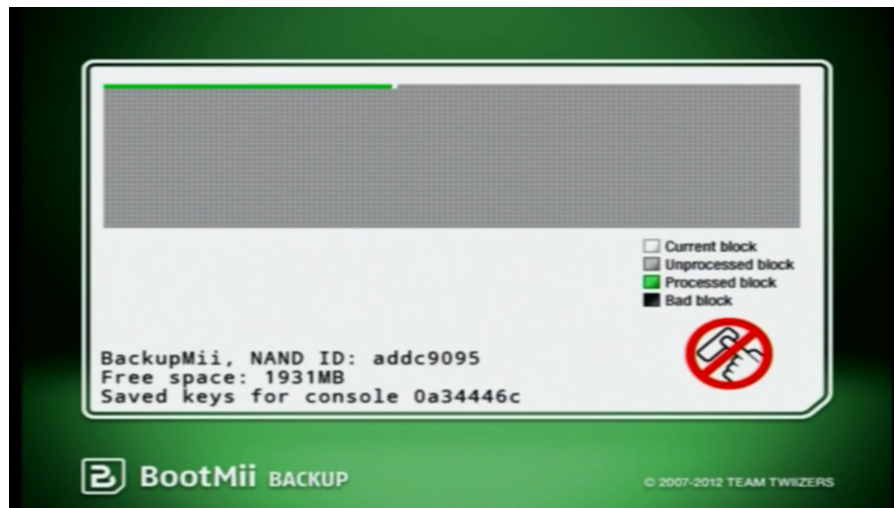


Figura 7.2: Screen del processo di backup della NAND Wii

7.3 Estrazione del file 00000011.app

Il certificato radice Nintendo assieme al certificato della console Wii e la sua chiave privata sono contenuti all'interno di un file della NAND della console Wii, chiamato *00000011.app*.

Il dump della NAND è cifrato tramite l'algoritmo **AES**² e quindi bisogna decriptarlo per estrarre il file citato dal suo interno. A tal fine si può usare l'applicazione *NandExtract*. Una volta avviata questa applicazione, si può caricare in essa la NAND tramite l'opzione *Open NAND...*, sotto la voce *File*. L'applicazione eseguirà la decifrazione, mostrando tutti i file e le cartelle contenuti dentro il dump caricato. Il file si trova nella cartella */title/00000001/0000000d/content* e si può estrarre cliccando su di esso con il tasto destro e successivamente con il sinistro utilizzando l'opzione *Extract*.

²AES: Advanced Encryption Standard, conosciuto anche come Rijndael, è un algoritmo di cifratura a blocchi a chiave simmetrica.

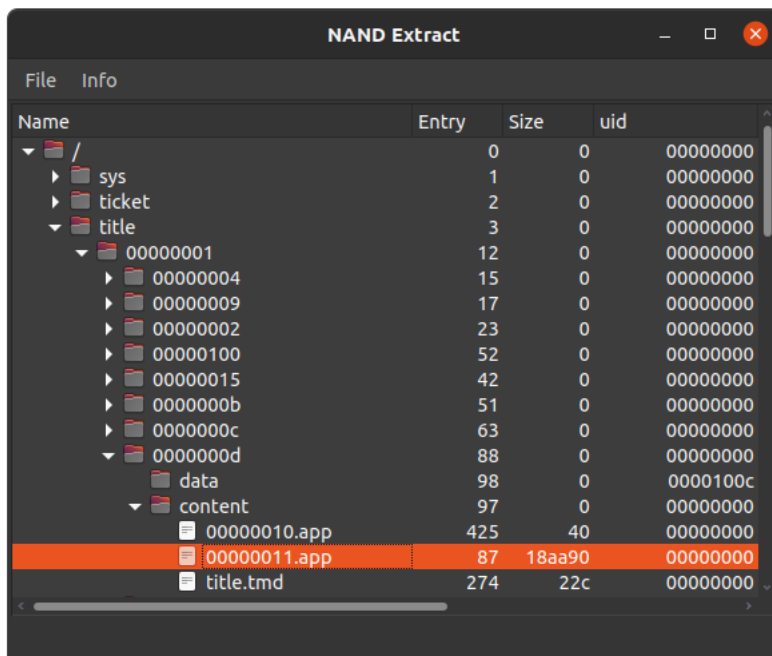


Figura 7.3: Screen dell'applicazione NAND Extract

7.4 Esecuzione dello script

Per poter utilizzare lo script, bisogna posizionare nella sua cartella il file *00000011.app* appena estratto. Successivamente bisogna raggiungere tramite terminale il percorso in cui si trova lo script e poi usare il comando `./nintendo_dwc_installer.sh` per eseguirlo. Nel caso in cui venga eseguito senza permessi di root, lo script chiederà all'utente di inserire la password per poterli impostare.

7.5 Utilizzo del server

Una volta che lo script avrà terminato, il finto server Nintendo sarà pronto a ricevere le richieste di connessione da parte delle console Nintendo DS. Nel caso si rifiuti la richiesta dello script di creare il *cron job*, bisognerà avviare manualmente *nginx* e *dnsmasq* assieme all'emulatore del server, come descritto nelle sezioni 6.9, 6.11 e 6.10, ogni volta che il sistema viene riavviato. A questo punto non resta che collegare la console al WiFi, modificare gli IP dei server DNS nella configurazione della connessione, come spiegato nella sezione 5.1, e infine connetterla al server appena creato. Se più console si connettono allo stesso server, queste potranno giocare insieme, comunicando tra loro.

In Figura 7.4 viene mostrata la connessione di una console Nintendo DS al finto server Nintendo, creato mediante l'utilizzo dello script descritto in questa tesi. Il server

mostrato è in esecuzione su un VPS³, a cui si è connessi tramite SSH⁴ e sul quale gira Linux Debian come Sistema Operativo.

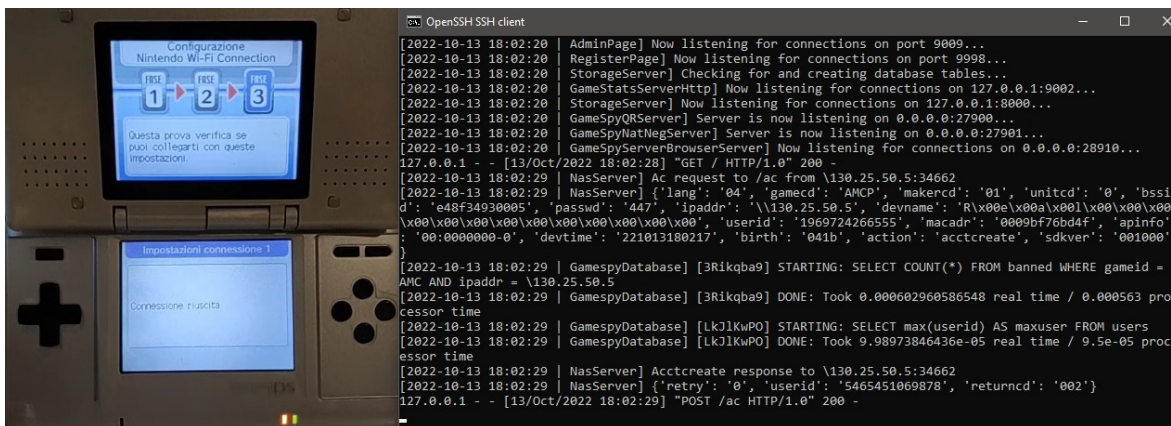


Figura 7.4: Screen della connessione della console al server

³VPS: Virtual Private Server, macchina virtuale che fornisce risorse server virtualizzate su un server fisico condiviso con altri utenti.

⁴SSH: Secure Shell, protocollo che permette di stabilire una connessione sicura ad un PC remoto, tramite internet.

8 | Conclusioni

In questa tesi è stata descritta una falla di sicurezza nella console Nintendo DS, dimostrando come questa possa essere sfruttata, utilizzando un bash script, di cui è stato mostrato il codice, per costringere la console a connettersi ad un server non ufficiale e consentirle quindi di giocare online, nonostante il servizio *Nintendo Wi-Fi Connection* sia cessato. L'operazione finalizzata a tale scopo viene definita attacco di tipo *WEB Spoofing*.

Per mezzo dello script, è stato possibile installare, in una VPS, un emulatore del server Nintendo implementato in python e aggiungere ad esso il supporto per SSL tramite *nginx*, in modo che il server potesse instaurare una connessione HTTPS con la console, mandandole un certificato da essa considerato valido, e potesse quindi gestirne le richieste. In ultimo, sono state connesse al server due console Nintendo DS, avviando una partita online.

La falla descritta è dovuta all'errata implementazione, da parte della Nintendo, del controllo della catena di certificazione, eseguito dalla console quando questa deve verificare il certificato inviato dal server a cui si sta connettendo. Il problema quindi non sta nel protocollo di sicurezza SSL, nonostante questo sia ormai deprecato. Ciò deve farci riflettere sul fatto che, nonostante esistano strumenti che garantiscono varie proprietà di sicurezza, il loro errato utilizzo può comunque comportare, in qualsiasi sistema, falle di sicurezza come quella descritta in questa tesi.

In conclusione, vorrei accennare ad un'idea che porterebbe ad ulteriori sviluppi di questo lavoro, ovvero aggiungere un'interfaccia grafica al server, al momento non implementata, sfruttabile tramite browser. Questa potrebbe mostrare dati statistici sulla connessione e quindi rendere più facile la gestione delle console connesse, impostando un limite massimo consentito o persino limitandone l'accesso.

Bibliografia

- [1] Svetlin Nakov. *Practical Cryptography for Developers*. SoftUni (Software University), 2018. ISBN: 9786190008705.
- [2] James F. Kurose Keith W. Ross. *Reti di calcolatori e internet - Un approccio top-down*. Pearson, 2017. ISBN: 9788891912282.

Sitografia

- [3] NGINX Docs. *Installing NGINX and NGINX Plus*. URL: <https://docs.nginx.com/nginx/admin-guide/installing-nginx>.
- [4] Simon Kelley. *DNSMASQ*. 2021. URL: <https://thekelleys.org.uk/dnsmasq/docs/dnsmasq-man.html>.
- [5] OpenSSL. *OpenSSL commands*. URL: <https://www.openssl.org/docs/man1.1.1/man1>.
- [6] D. Cooper S. Santesson S. Farrell S. Boeyen R. Housley W. Polk. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. URL: <https://www.rfc-editor.org/rfc/rfc5280.html>.
- [7] Real96. *Nintendo-SSL-DWC-Installer-Script*. 2022. URL: <https://github.com/Real96/Nintendo-SSL-DWC-Installer-Script>.
- [8] shutterbug2000. *nds-constraint*. 2018. URL: <https://github.com/KaeruTeam/nds-constraint>.
- [9] SSL.com. *Archivi: domande frequenti*. URL: <https://www.ssl.com/it/faq>.
- [10] Vega. *Tutorial to Setup Your Own Local MKW Server*. 2018. URL: <https://mariokartwii.com/showthread.php?tid=885>.