**INTERNET SECURITY**

L’esame verterà su un progetto (SW + relazione), senza voto e unico. Solo dopo averlo superato ne conseguirà un colloquio orale; nel caso in cui non si superi la prova orale, il progetto rimane comunque valido per le successive prove orali per tutto l’anno accademico, cioè non si deve rifare. Consigliato il testo di Stallings.

Progetto assegnato: A02-2021 Cryptographic failures. Trovare due o tre esemplificazioni in attacco, da OWASP.

**LEZIONE 1**

**INTRODUZIONE**

Con l’evoluzione della tecnologia, i problemi legati ad essa si evolvono e cambiano nel tempo. Per esempio, attualmente vi è il problema del **Double Spending**, dovuto alle monete virtuali (sono dei file) in app, il quale consiste in una truffa tramite la quale si riesce a spendere lo stesso titolo valutario due o più volte. Quanto codice facciamo girare ogni giorno senza farci caso, e quanti **atti di fede** facciamo sperando non ci sia nulla di malevolo? La fiducia viene riposta sia su SW che su HW.

**Che nesso c’è fra cybersecurity e fiducia riposta in essa?** Per quanti sforzi possiamo fare, in qualsiasi contesto e con qualsiasi soluzione una dose di fiducia si dovrà sempre avere. Per esempio, RSA è un algoritmo crittografico che funziona bene, ma sempre sotto buone assunzioni e ipotesi, come quella di randomizzare bene o scegliere numeri primi abbastanza grandi, altrimenti il sistema sarebbe raggirato. Le varie vulnerabilità vengono messe su determinati repository, e seppur si pensi che Apple essendo un sistema chiuso potesse avere meno problemi, così non è, è al pari con Windows.

**Qual è il nesso tra una soluzione gold e soluzioni alternative per fixare una falla nel sistema?** Fare fix SW è molto complesso, perché bisogna prendere in considerazione molte cose: se il SW funziona dopo il fix, se necessito di cambiare SO, quanto devo pagare chi sviluppa il fix, se è necessario stoppare la continuità del servizio. In sintesi, bisogna fare un’analisi costi/benefici, mettere addirittura in discussione il senso stesso del fix. Ci sono infatti libri di Security Economics che affrontano questo problema. Nel mondo reale del lavoro capita spesso che non si ha il budget per fare il fix, quindi bisogna fare delle rinunce, bisogna cercare delle alternative e soluzioni più applicabili, per questo serve creatività, flessibilità e competenza. L’aggiornamento del SW è la prima misura di sicurezza, è il meglio che si può fare, è l’opzione che aumenta di più le possibilità di fissare le vulnerabilità accettandola per fede. Anche qui bisogna fare un bilancio sicurezza/prestazioni, poiché spesso capita che il sistema aggiornato sia un po' più lento, ma comunque rimane molto consigliato aggiornare. Anche negli aggiornamenti, durante la loro distribuzione (rollout), bisogna valutare se si deve interrompere la continuità del servizio, e in alternativa avere ridondanza di server (DR). È sempre molto importante il bilancio funzioni/sicurezza.

**ESEMPI DI PROBLEMI DI SICUREZZA**

C’è un’infinita **varietà dei sistemi in cui serve la sicurezza** ed è fondamentale il **nesso tra attacco e policy.**

I problemi di sicurezza sono dappertutto, in qualsiasi risorsa che possa avere un valore di qualsiasi tipo e che possano essere di interesse per un attaccante, anche una foto o un video.

I **DB**, ricchi di dati, sono spesso oggetto di attacco, e agli attaccanti interessa come i dati sono programmati. Che problemi si hanno nell’ambito sicurezza di un DB?

* 1 – DBMS buono, ma i dati stanno su disco, e posso attaccare facilmente il sistema portando i dati dal disco; per questo è necessaria una tecnica **TDE** (Transparent data encryption) che ci permette di cifrare i dati.
* 2 – L’utente riesce a vedere più di quanto dovrebbe, quindi c’è un problema di autenticazione.
* 3 – Il DB è ben progettato, ma esposto a web-app che permettono agli utenti di inviare query malformate (di proposito malevole), e per esempio fare il drop di tutti i dati, o danneggiarli. Un ulteriore livello di sicurezza si può aggiungere con tool che pseudonimizzano, generalizzano, pseudotipizzano le tabelle, e consistono per esempio nello scrivere xyz invece di Mario Rossi, per irrobustire il DB.

Per quanto riguarda la sicurezzadei **Sistemi Operativi**, se uso una macchina in condivisione con altri utenti, in teoria l’utente 1 non dovrebbe vedere lo spazio disco degli altri, anche se in realtà tutto è deciso dalle policy. La policy dei SO viene decisa da chi fa la distro, quindi vi può essere molta varietà, e da essa dipende la sicurezza, cioè la coerenza del sistema con le sue regole di funzionamento. Si parla tecnicamente di attacco o meno, in base a ciò che è concesso dalla policy, è essa a circoscrivere il recinto. Cioè, se un utente (non root) fa ls sullo spazio di un altro utente, in teoria non dovrebbe vederlo, ma nel caso contrario in cui lo veda, non si tratta di un attacco, perché è la policy a concederlo. Se si va contro la policy allora è un attacco. La sicurezza è un problema di legge. Attenzione che la policy vieta sempre di poter scrivere nello spazio di altri utenti, perché è un’azione pericolosissima.

Ormai ad oggi tutti i SW devono avere la necessità di essere protetti, si parla di **Security by Design**, ormai a priori affiancato al progetto del SW vero e proprio. La sicurezza è radicata in ogni ambito della vita ormai: guerra, politica, e per ogni attacco diverso servono competenze specifiche.

I protocolli qui affrontati dovranno permettere di cifrare e nascondere ciò che passa tramite i protocolli di rete, cioè ogni tipo di messaggio scambiato tramite internet e molto altro. Il problema di fare le cose sicure è arrivato infatti dopo che le cose funzionassero, la sicurezza è un requisito non funzionale infatti. Paradossalmente questi sistemi di sicurezza, gli algoritmi crittografici, sono pubblici, si possono apprendere e conoscere, poiché il fulcro di tutto sta nella conoscenza della chiave, che rende il sistema forte contro l’attaccante.

Un’intrusione è un processo alieno.

**TROJAN**

In genere col nome **Trojan**, si indica un particolare tipo di malware che prende il controllo, parziale o totale, del sistema attaccato. Esso si presenta alla vittima come un programma, un file esca, incitando in qualche modo la vittima ad usarlo, e di conseguenza esso stesso lo attiverà. È solo un esempio di codice malevolo; altri potrebbero essere back door, logic bomb eccetera.

Ogni file ha il proprietario e un gruppo, il primo ha tutti i permessi su esso. Normalmente un programma riceve i permessi dell’utente che lo lancia. **Passwd** (in etc) è un programma che appartiene a root, il quale permette di modificare la password. Il problema è la scalabilità nel caso in cui debba essere root a dover cambiare la password di ogni utente, e inoltre non è sicuro perché egli conoscerebbe tutte le password degli utenti. Quindi come potrebbe un normale utente modificarsi la password? Si dovrebbe poter permettere all’utente, per un attimo, di avere “poteri root” e fare l’operazione di cambio password. È un’idea progettuale del mondo Unix, e si ha tramite il **bit S o SUID (Set owner User ID),** tramite il quale l’utente segnala la volontà di chiamare passwd, e verrà lanciato con i permessi dell’owner (root in questo caso). La sicurezza della procedura consiste nel richiedere la vecchia password prima di poter metterne una nuova, quindi c’è autenticazione. Così si bilancia il lato “attaccabile” di SUID, sfruttando la sicurezza del codice di passwd. È sempre importante il **bilancio tra funzione e sicurezza**. Cosa succederebbe se si creasse una shell (da attaccante) con il bit SUID?

COMANDI SHELL

su piero -> provo a entrare con utente piero (anche root)

userdel piero -> elimino piero

**LEZIONE 2**

**ESEMPIO DI TROJAN HORSE**

Il Bit S è una soluzione storica, usato anche per altro. Quando esso è attivo su passwd, significa che gli utenti possono chiamarlo e cambiare la propria password. Permette quindi una privilege escalation, cioè un aumento di privilegi controllati. Potenzialmente è un aspetto vulnerabile, può essere sfruttato per costruire un Trojan, e tecnicamente si parla di **attacco alla privilege escalation**.

Immagine che contiene testo, lettera

Descrizione generata automaticamente

Questo codice è il contenuto di un trojan, per fare ciò esso **deve essere eseguibile**, dobbiamo renderlo tale poiché un file appena creato per policy non lo è, e in qualche modo lo facciamo arrivare alla vittima e glielo facciamo eseguire. Ovviamente **c’è qui una componente umana**, cioè la vittima deve accorgersene per sviare il problema. Nelle ultime versioni di Linux questo Trojan non è però funzionante perché fixato, in particolare è stata cambiata la policy di cp che dà al file i permessi della directory su cui è copiata, non potendo quindi avere il bit S.

**Tramite la prima linea di codice**, prendiamo l’eseguibile classico della shell (tramite questa possiamo fare molto, è abbastanza potente) e lo mettiamo in un nuovo file nella cartella /tmp e gli diamo il nome .xxsh che in questo caso è un file nascosto (cioè per esempio creando un file .trojan.sh, abbiamo creato un normale file trojan.sh che però è nascosto, serve fare ls -a per vederlo). Abbiamo scelto di mettere il nuovo file .xxsh in tmp perché siamo sicuri che la directory nella vittima esiste (anche home c’è sempre, ma in tmp è un po' più nascosta, è meno frequente che si usi). Lo faccio su una copia della shell (sh di bin) perché prendere il possesso della shell vera e propria non si può. Inoltre, si sfrutta la policy secondo la quale su bin, per questioni di permessi, noi attaccanti (other) non possiamo fare nulla, mentre in tmp possiamo settare i permessi (other) come vogliamo. Ovviamente bisogna capire come cp copi i permessi nel nuovo file: in genere gli dà i permessi della directory in cui lo copio, o potrebbe anche dare i permessi del file originario (è una scelta). Quindi per policy del file-system si intende non solo la compartizione degli spazi e i permessi, i proprietari, ma anche che permessi danno certi comandi come in questo caso cp. In bin solo root può scrivere, quindi non posso piantare il bit S, mentre su tmp posso scrivere e quindi piantare SUID.

**Tramite la seconda linea di codice** cambio i permessi del file appena creato (il trojan): dà il bit S all’owner (u+s) e nel dubbio(!) l’eseguibilità a tutti gli altri (o+x). Ho quindi fatto una shell con il bit S e che tutti gli altri possono eseguire. Glielo facciamo eseguire con il bit S. (quindi posso cambiargli la password?). Insieme alla prima rappresenta il payload o carico, il succo del trojan.

**La terza linea cancella** ls dalla cartella corrente, cioè il trojan stesso che abbiamo chiamato appunto “ls”, per non lasciare le tracce (“ls” della cartella corrente è differente da ls del $PATH). Autodistruzione.

**La quarta linea** mostra all’utente l’esecuzione della chiamata di sistema, per bluffare l’utente facendogli credere che tutto funzioni correttamente, quando in realtà abbiamo fatto danno e cancellato le tracce.

L’attacco vero e proprio si ha quando, dopo che la vittima ha eseguito ./ls, .xxsh sta adesso in /tmp della vittima. (ricordare che tmp e bin sono al pari di home, cioè, sono condivise tra i vari utenti). Quindi l’attaccante esegue /.xxsh con i privilegi di victim, e adesso avendo una shell sullo spazio dell’altro utente, può fare tutto soprattutto scrivere (se adesso facessi passwd?). Per vedere se tutto è andato a buon fine, fare **whoami** su shell attaccante e dovrebbe essere victim.

Questo trojan funzionava perché **“.” stava nelle** **variabili di sistema**, cioè $PATH, che è una lista di percorsi in cui andare a controllare se c’è il comando eseguito. Questo perché prima tutto era nella home, ed era comodo che . fosse nel PATH, addirittura il primo tra i path in cui scegliere (mai avere il punto nel PATH e metterlo come primo soprattutto, poiché si aumenterebbe il rischio che qualcosa che capiti nella home venga eseguito con priorità superiori ai programmi benigni del sistema). Quindi scrivendo “ls” avrebbe eseguito per priorità “./ls” invece del comando generico “ls”.

Adesso il . non è più nel PATH quindi la vittima avrebbe dovuto scrivere esattamente “./ls”, indebolendo molto la possibilità di attacco.

**Come si potrebbe mitigare un trojan scaricato dalla rete?**

* Togliamo dalle variabili di sistema ($PATH) la cartella del quale il browser di default scarica il file, se presente perché, se è nel PATH partirà prima di ogni altra cosa (rischioso e incognito);
* Dire al browser di dare all’utente il solo permesso di lettura, così egli può leggerlo e verificare effettivamente che si tratta di qualcosa di benigno, allora egli stesso potrà tramite chmod settare eseguibilità o scrittura. Come notiamo infatti, alla creazione di un file generico (il trojan in questo caso), bisogna conferirgli l’eseguibilità tramite chmod dopo averlo creato, di default non la ha.

In conclusione, il proprietario della shell copiata è la vittima. Chiunque andrà a eseguire la shell, avrà i suoi permessi e quindi potrà leggere e scrivere sullo spazio della vittima. Per quanto riguarda passwd tramite shell malevola, se la vittima fosse l’admin, dalla shell malevola l’attaccante potrebbe fare passwd victim e aggirare il controllo password prima di inserire quella nuova, e quindi cambiarla a chi vuole. Attaccando invece un normale utente, ciò non accade: innanzitutto ci fa cambiare la password dell’attaccante, e supponendo così non fosse, ci richiederebbe la vecchia password.

**LEZIONE 3**

**VIOLAZIONI RECENTI**

Vulnerabilità simili a questo trojan ce ne sono infinite. Può esserci addirittura su Youtube, ma su qualsiasi altra piattaforma “affidabile”, **pubblicità avvelenata**, malevola, perché bisogna ricordare prima di aprirle che contengono codice, il quale viene eseguito in maniera del tutto trasparente dall’utente, e quindi può essere malevolo, potrebbe anche portare a un URL cattivo. Quindi un click attiva tante cose di cui non ne siamo a conoscenza, e si fanno continuamente atti di fede senza accorgersene. **Anonymous** è un gruppo di criminali per cui è motivo di gloria piantare i sistemi. Spesso fanno attacchi DDoS, cioè, interrompono un servizio sovraccaricandolo con traffico dannoso. Un altro aspetto da non sottovalutare, è la presenza di **bloatware** nei dispositivi che acquistiamo, cioè SW già presente all’acquisto senza richiesta del consenso dell’utente; ovviamente si pensa che sia sicuro, ma non sempre è così: di recente se n’è scoperta una che permetteva il tracking persistente dell’attività utente. Attenzione quindi anche alla fiducia riposta al SW preinstallato.

**STRUMENTI DI SICUREZZA**

Così come ci sono innumerevoli possibili falle nel mondo tecnologico, d’altra parte vi sono innumerevoli tentativi di difesa. È un gioco tra parti in continua evoluzione e che ha come unico limite quello economico (i pc quantistici esistono ma sono comunemente troppo costosi).

La cifratura, **crittografia**, è uno degli strumenti fondamentali, costa tanto sia in termini economici che umani, poiché necessita molto lavoro e preparazione. Solo da pochi anni si sta iniziando a pensare di criptare tutto by default, e si dovrebbe mirare a un totale criptaggio. Essa può essere simmetrica (chiave condivisa) o asimmetrica (chiave pubblica).

Le **policy** abbiamo già visto come siano importanti per un sistema sicuro, sono ciò che ci permettono di discernere, distinguere tra ciò che è lecito e ciò che è illecito. Avere una policy ovviamente non implica che tutti la seguano.

La **conoscenza** è anch’essa uno strumento di sicurezza perché, se password/pin è conosciuta solo dal proprietario, è un modo di essere più sicuri, o avere il possesso fisico di un biglietto. Anche la biometria è importante perché si basa su caratteristiche uniche che ognuno di noi ha, come le impronte o l’iride.

Poi si hanno anche **programmi di protezione** come IDS (Intrusion Detection System) e firewall, i quali si comportano come i portieri in un condominio, hanno una loro policy e in base a questa decidono chi può entrare e chi no. Il problema del firewall è che non interroga come il portiere, ma guarda solo indirizzo di arrivo e di destinazione. La differenza con IDS è che quest’ultimo guarda come siamo fatti, il contenuto.

Per quanto riguarda i **protocolli di sicurezza**, mentre Telnet è un applicativo per accesso remoto ma manda login e password in chiaro, SSH lo manda cifrato. SSL invece è un protocollo che applicato ad HTTP ha portato ad HTTPS.

Non da sottovalutare e di uguale importanza è la **sensibilizzazione dell’utente**, cioè il fattore umano. Le persone devono essere istruite e informate sui tanti problemi e devono di conseguenza agire con cautela nell’utilizzo di macchine, che siano PC o telefoni. Per esempio, le e-mail, come nel contesto precedente di poisoned URL, arrivano in html; quindi, potrebbe esserci dentro codice malevolo e fare danno una volta aperta la mail. Per evitare ciò si potrebbe disattivare il caricamento automatico della mail disponendo di una preview non in html, per guardare il mittente e solo allora nel caso in cui sembri affidabile, fare interpretare in html (è una tecnica anti-fishing). Il fishing è una pratica che consiste nell’ingannare la vittima fingendosi un’entità, al fine di rubare dati.

In sintesi, si può però affermare che la sicurezza assoluta non esiste, anche combinando tutti o alcuni di questi strumenti. Nelle aziende si usa un approccio **risk based**, cioè si valuta il rischio, che ovviamente non si riesce mai ad azzerare.

La crittografia è una scienza esatta come la matematica. È basata infatti su dimostrazioni con ipotesi e la debolezza sta proprio qui, poiché esse potrebbero essere aggirate o non applicabili. La sicurezza invece non è una scienza esatta.

**PASSWORD**

Un attacco storico alle password è **l’attacco dizionario**. L’utente tende a mettere password mnemoniche e lo ha fatto soprattutto nei primi tempi, dal 95 al 2005. Il dizionario è composto da 250.000 parole che potremmo reiterare e provare nelle password. È un attacco primordiale. A poco a poco si capì che bisognava aumentarne la sicurezza, e si ampliò il campo di scelta sulle password combinando i caratteri con numeri o simboli. D’altra parte, anche gli attaccanti si rifornirono di un vocabolario più complesso, e qui si nota il perenne scontro tra attacco e difesa. Il bilancio (plateux) si raggiunge non per motivi tecnologici, basti pensare ai calcolatori quantistici, ma per questioni economiche. L’attacco dizionario moderno ha quindi varianti del dizionario. Quindi dove sta il limite? Dato che l’attaccante potrebbe, ipotizzando abbia tempo e potenza necessaria, **brute-force-are** ogni password, si è pensato di inserire il **controllo a soglie**, tramite il quale è possibile provare per un singolo account un numero limitato e molto basso di password. Delle possibilità in più sono nelle mani degli attaccanti, quando questi conoscono qualcosa per via diretta o indiretta (per esempio tramite social o osservando la vita della vittima), e cercano di stringere il campo del vocabolario, cioè, aumentano le probabilità di beccare la password giusta. (per esempio, la vittima è appassionata di coccodrilli o nuoto, ha una certa tendenza a fare una cosa). Questo è il caso degli **attacchi statistici** che sfruttano tecniche di **social engineering**.

**Come faccio a scegliere una buona password?** Essa ovviamente deve essere il più lunga possibile, ma d’altro canto deve essere mnemonica per essere ricordata. È importante anche la periodicità del cambio password, cioè ogni tot mesi si cambia (oggigiorno non è permesso addirittura inserire password simili a quelle precedenti). Questo è un protocollo di sicurezza, ma bisogna sempre considerare la bilancia sicurezza-funzionalità; infatti, la difficoltà sta poi nel ricordarla, poiché oggigiorno si hanno moltissime password. Si tende spesso ad inserirle uguali o simili nelle varie piattaforme/app; infatti, gli attacchi recenti si basano su questo riuso. Anche il fatto di reimpostare la password in caso si dimentichi è un protocollo di sicurezza, e deve essere forte, infatti richiede innanzitutto la password vecchia per metterne una nuova, sennò sarebbe molto vulnerabile. Ma anche qui si fa un grande atto di fede, perché il form di reset password viene spesso inviato via e-mail, su cui bisogna quindi fare grande affidamento (oggi, infatti, si è irrobustita dal punto di vista della sicurezza).

In conclusione, come si fa una password robusta? **Bisogna bilanciare al meglio la difficoltà di indovinarla con la mnemonicità.** Il NIST, istituto americano degli standard, nel 2004 diffuse l’idea di mettere una password con un determinato formato, con una certa lunghezza, caratteri speciali e numeri. Ha rivisto poi la propria posizione nel 2017, dicendo di usare una password semplice con un controllo sulla soglia di possibili inserimenti, in modo da non essere brute-force-able. Sono però delle linee guida che necessitano molto tempo per essere applicate in tutti i sistemi, è stata una scelta anche molto criticata. Quindi qual è la migliore password? È una domanda difficile.

**LEZIONE 4**

**FIREWALL**

Il firewall è solo una **difesa perimetrale** tra ambiti con autorizzazioni di sicurezza diverse (per esempio l’università lo usa per circoscriversi). Posso anche avere firewall tra docenti e studenti dentro la difesa perimetrale universitaria. **L’insider** però ci può sempre essere, ciò significa che non importa quanto il firewall circoscriva. A meno che non si parla del firewall personale del proprio pc (UFW), il problema sussiste sempre. Il firewall non ha capacità di guardare il contenuto, ma solo indirizzo di arrivo e destinazione.

**DEFINIZIONE DI SICUREZZA**

1. Non è un prodotto, ma un **processo** (macro). A differenza di tool, SW, SO, che sono prodotti e si possono studiare dall’inizio alla fine, la sicurezza non ha questi due estremi, ma è una serie di procedure che vanno eseguite in ciclo continuo, si evolvono e cambiano nel tempo. Infatti, se ci fosse uno step di formazione del personale sulla sicurezza, dovrebbe essere fatto in maniera perenne. Ci sono delle best practices, linee guida che spiegano come assicurare la continuità operativa, anche contro attività malevole, una di questa è la procedura **Incident Response**, scritta nell’ISO, la quale spiega come comportarsi in seguito ad un attacco. Esistono anche procedure pre-attacco, per evitare che avvenga. In genere si necessita un sito di DR (Disaster Recovery).
2. È fondamentale **l’anello più debole della catena**. È un concetto astratto. Se ho un wi-fi locale, e un pc debole si connette ad essa, sta qua l’anello debole. Cioè, posso anche avere una casa super sicura, ma se poi ho una lampadina smart tutta la sicurezza salta (vedere problemi lampadina smart).
3. Devo sempre aggiungere il complemento “da che cosa”, cioè **sicurezza da chi o cosa**? Sempre per il fatto che non si ha sicurezza assoluta.
4. È un **gioco perpetuo di irrobustimento difesa e attacco**, il quale terminerà quando l’analisi costi/benefici va in negativo, cioè quando non ne vale più la pena difendere o attaccare perché anche in caso di esito positivo, si perderebbe. Ciò evidenzia anche il “da che cosa” siamo sicuri del punto 3.
5. È bene evidenziare **vari livelli di sicurezza**, distinguendo da quali attaccanti si ha la difesa e da chi invece non si riuscirebbe a reggere un attacco.

**RISCHI BASE PER LA SICUREZZA**

* Il browser è per esempio il **sistema** più **complicato** in assoluto. Più un sistema è complicato, più è difficile da mettere in sicurezza.
* I sistemi interagiscono tra di loro quasi sempre, basti pensare al nostro wi-fi. Appena avviene questa **combinazione**, la **complessità aumenta**, perché bisogna che tutti siano sicuri e che non ci siano punti deboli nella catena. Si potrebbe quindi non preservare la sicurezza individuale del sistema. Per esempio, se il trasporto delle password del wi-fi è in chiaro, tramite lampadine intelligenti hackerabili che permettono l’intercettazione.
* **Predisposizione ai bug**: è una proprietà inattesa e quindi sinonimo di vulnerabilità. La differenza tra bug e feature sta nell’intenzionalità, cioè, dipende dalla policy. Il bug è un qualcosa che sfugge dal normale controllo, non viene considerato e quindi sistemato. La feature è invece una caratteristica implementata di proposito.
* **Proprietà emergenti**: se inventiamo sistemi nuovi ci saranno problemi nuovi, come per esempio il double spending o l’esempio della valigia virtuale che si poteva fare online ed essere spedita dove si vuole.
* **Interazione con l’uomo**, che è sempre il mezzo principale da cui tutto scaturisce, con una buona probabilità è egli l’anello debole della catena.

**RISCHI DIGITALI, PERCHE’ LA SICUREZZA È UN PROBLEMA?**

Essi sono tipici perché riguardano il mondo digitale. Per quanto riguarda l’automazione offensiva, essa si ha perché, se un attacco funziona una volta lo posso reiterare su ambienti simili, quindi sono facili da automatizzare. Per quanto riguarda l’assenza di distanza invece, essendo esposti in rete, siamo potenzialmente vicini e contro tutto il mondo, poi subentrano anche problemi di territorialità con confusione in caso di attacco (quale policy e leggi considero?). Vi è anche una facile propagazione delle tecniche di hacking, con facilità anche di utilizzo scaricando script o SW automatizzati. Gli adolescenti sono i più colpiti da questi attacchi in quanto hanno una percezione di rischio molto bassa. Per quanto riguarda la reazione è molto difficile, l’incident response è complicato. Per esempio, l’IDS rileva un processo alieno, e killandolo non risolve la situazione perché potrebbe rispuntare. Bisogna risolvere alla radice vedendo la causa della vulnerabilità, il punto di accesso di esso (exploit), e non è per nulla semplice.

Il gioco della sicurezza è un continuo loop, coerentemente col fatto che la sicurezza è un processo, non un prodotto. Vedere metodologia d’attacco e difesa con i vari step in loop (sommariamente). In genere il team rosso attacca e il blu difende.

**PORTE DI SISTEMA**

Tutto ciò è nato a scopo buono, ma poi le diagnostiche si sono rilevate utili per gli attaccanti. Port: È un indirizzo di rete implementato dal SO che aiuta a distinguere il traffico destinato tra le differenti applicazioni e servizi. Ci sono delle porte standard. Il Port Scanning è una diagnostica utile all’attaccante, per conoscere il sistema target e vedere le sue possibili debolezze. Si basa sull’accesso alle porte disponibili per vedere quali sono aperte, e acquisendo informazioni su quali servizi ci stanno dietro quella porta. La scansione (nmap), possibile con tante opzioni e tanti flag, è come il ladro che vuole sapere chi sta in casa prima di rubare. Facendo lo scanning vuole capire chi c’è dietro la porta senza farlo capire a chi c’è dietro la porta, poiché in caso contrario quest’ultimo si munirebbe di difese. Si parla infatti di scansione stealthiness. Facendo il port scanning, capisco il tipo di difensore con le sue vulnerabilità, e faccio un attacco di conseguenza. Non serve il port scanning per attaccare via web, bensì per attacchi di rete.

Come fa nmap ad essere stealth? Tramite la frequenza di traffico e il binding, su come setta l’handshake. L’handshake è la “stretta di mano” tra due processi che comunicano, i quali sono d’accordo sul loro modo di dialogare. In TCP lo scambio è a tre vie, chi fa la scansione può quindi farla a uno o due vie. Se l’handshake è completo, nmap è potente e può trovare tutto; viceversa, se non è completo, potrebbe essere qualsiasi cosa con falsi positivi. In sintesi, più l’handshake termina precocemente, minore sarà la possibilità che l’attaccante sia scoperto, ma minori saranno le informazioni che potrà trovare.

L’IDS normalmente non è integrato col SO. C’è il rischio di avere anche tanti falsi positivi, non c’è un modo assoluto per identificare un’intrusione.

**LEZIONE 5**

**ATTACCHI**

A seconda di ciò che si attacca e come, vi è una classificazione. Si potrebbe per esempio fare un attacco per denaro, rubare risorse computazionali, piantare servizi, attacchi reputazionali, attacchi effort (si attacca entrando tramite un dipendente “debole”). Per attacco si intende un’attività malevola per violare un sistema. La sicurezza va di pari passo con la qualità del servizio, ed è appunto un requisito non funzionale, ma ciò non implica che non sia fondamentale. I requisiti funzionali sono invece ciò che il SW fa. L’obiettivo è appunto quello di garantire un servizio funzionante e sicuro, perché sarebbe facile garantire il servizio spegnendo il server per questioni di sicurezza, ma non si può.

Contromisure dal punto di vista teorico: l’attaccante ha il dispositivo e cercherà di accedere al sistema, la nostra contromisura è la misura di autenticazione, password o biometrica; l’attaccante potrebbe anche stare cercando di accedere alle funzionalità del sistema, o i dati, o entrambi. Per questo è fondamentale la password del BIOS, la quale permette sicuramente di non accedere alle funzionalità, ma non garantisce la sicurezza dei dati, in quanto le memorie secondarie sono spesso rimovibili. Dopo che l’attaccante ha avuto accesso, vuole usare il sistema stand-alone, su locale (cioè, indipendente e isolato da altri pc), ma soprattutto cerca funzionalità di rete, dato che al giorno d’oggi tutto sta su cloud. La contromisura sta nelle ulteriori misure di sicurezza che l’attaccante incontrerà, cioè password per applicativi vari (e-mail, cloud). L’attacco successivo possibile sarà quello di acquisizione di dati sensibili, e qui la contromisura è la crittografia dell’intero file-system, compreso il boot, o solo della home (attenzione che, se non ho password in BIOS la macchina parte se cambio hard-disk).

Nella pratica però succede tutt’altro. L’attaccante può aggirare l’autenticazione al sistema avviando da una USB, facendo la mount dell’unità locale, dopo di che vedrà tutto lo spazio e potrà copiarselo. Qui entra in gioco la policy del SO, perché montando un’unità impongo delle regole assolutamente a scelta, potendo addirittura vedere nello spazio di root. Quindi le password del sistema non garantiscono sicurezza. L’avvio da USB era in realtà nato a scopo benevolo, per ripristinare in caso ci si fosse dimenticata la password.

Per quanto riguarda le funzionalità principali, esse nella realtà sono senza autenticazione, basti pensare all’accesso a un browser, poiché le principali attività vengono fatte da lì. Firefox ha una master password, che permette a sua volta di accedere alle password salvate nel browser, ma nessuno la conosce e usa. Edge invece è integrato nel SO di Windows, quindi si ha già la sicurezza integrata delle password tramite la password utente. Anche qui l’atto di fede consiste nel far salvare ai browser le password nel loro cloud, e nella tecnologia di salvataggio della password, affidandoci a qualcosa e qualcuno che non conosciamo. Ecco, comunque, che l’attaccante si impersona, dopo aver avuto accesso al sistema, ruba le password dal browser (di tutti i tipi). Può addirittura accedere senza scoprire le credenziali (accesso automatico, quando ci chiedono se vogliamo mantenere l’accesso). Master password ha una funzione duplice: se non inserita non permette di vedere le password del browser, e senza di essa non si possono usare le password. Senza inserimento di ri-autenticazione (che va contro l’usabilità) negli applicativi, l’attaccante bypasserà le password, e anche senza scoprirle arriverà al suo scopo (è un attacco molto ricorrente). C’è un trend verso l’usabilità a discapito della sicurezza, basti pensare agli acquisti di massimo 25 euro con le carte senza autenticazione, solo strisciando. Tutte le misure di sicurezza sono vane senza full disk encryption.

**ATTACCHI REALI**

La pirateria digitale col tempo sta diventando più rara, perché sta cambiando il modello di business, basti pensare a quanti prima crackavano i SO, e quanti adesso lo tengono crackato. Ha un climax, un crescendo: dal furto di proprietà intellettuale, al furto di identità (più grave) al furto di marchio (in quest’ultimo caso la difesa è la cyberlaw). Il watermarking è un artefatto grafico che marchia permanentemente un file, permettendo di riconoscerne uno originale da uno modificato. Però ci sono ormai algoritmi che cancellano i watermark ed è diventato un sistema di difesa fragile al pari della firma digitale. Lo sniffing è un altro tipo di attacco, in cui l’attaccante si interpone tra due utenti, e mettendosi nella loro stessa rete, linea, intercetta dati.

Si possono distinguere diversi tipi di attacchi. Gli attacchi criminali si hanno quando l’attaccante è un delinquente che vuole ottenere qualunque beneficio e il cui limite è solo la fantasia; essi sono partizionabili in più parti, come le frodi (rubare soldi), attacchi distruttivi e furti fari, tutto spesso tramite fishing. Le violazioni alla privatezza si hanno per esempio se tutti ricevono e vedono le buste paga di tutti gli impiegati. Gli attacchi a scopo pubblicitario, per esempio, si ebbero con le supposte elezioni di Trump pilotate dalla Russia, portano ad orientare la massa; un altro esempio opposto è quello del DoS, cioè, mandare giù un servizio con lo scopo di farle perdere fama. Gli attacchi basati su sistemi legali si hanno quando per esempio si manovra il parere dell’esperto in tribunale.

**PROPRIETA’ DI SICUREZZA**

Cos’è la sicurezza? È un insieme di proprietà di vari livelli,1,2 e 3. Quelli fondamentali sono alla base, l’ISO 27001 che tratta della sicurezza dei sistemi informatici usa l’acronimo CIA (confidenciality, integrity, avialability). C’è differenza col punto di vista del docente e le slide. È strano che non parlino di autenticazione nell’ISO, ma in realtà sta nascosta dietro il concetto di integrità.

**1° LIVELLO**

**SEGRETEZZA (CONFIDENZIALITA’)**

Il presupposto fondamentale è che ci sia una policy che sancisca chi debba conoscere e chi no. In un segreto, infatti, si cela una policy che regola ciò, discernere tra chi può e chi non può. Quindi la segretezza sta nel non rilasciare l’informazione a chi non è autorizzato. Si parla di segretezza di password, chiavi; in questi a conoscere il segreto si è in 2, il soggetto e il pc. Spesso, infatti, il segreto è conosciuto da almeno 2 entità. Come si può ottenere? Con la crittografia, che è un protocollo. La cifratura non nasconde, trasforma qualcosa in incomprensibile, non deducibile per chi non conosce la chiave. Con la steganografia (Chaffing & Winnowing) che rende indistinguibili le informazioni nascondendole, quindi ottengo segretezza certamente, ma la crittografia è più scalabile. Una tecnica steganografica è quella del Least-significant bit che consiste nel nascondere un messaggio nei bit meno significativi, il quale comporta anche confidenzialità. Tuttavia, è un algoritmo di encoding ormai debole, come il cifrario di Cesare per la crittografia. Ce ne sono altri molto robusti per garantire la segretezza. Esiste proprio una scienza, la crittoanalisi, che studia come rompere un sistema criptato; essa al contrario vuole trasformare in comprensibile qualcosa di crittografato, senza conoscere la chiave di crittografia.

**LEZIONE 6 (seminario Hardening)**

Oggigiorno le aziende che forniscono sicurezza, devono scrivere policy, le quali devono essere autorizzate; bisogna infatti dimostrare che tutto il personale dell’azienda ne è a conoscenza e lo applica. Si parla di best practices che bisogna applicare, per esempio quella di sensibilizzazione dell’utente per evitare il data breach (fuga di dati intenzionale o meno da un ambiente protetto). Ormai tutte le aziende in cui la componente digitale è importante, per norma devono avere la sicurezza integrata nei loro sistemi informatici, in quanto vigono norme di cybersecurity da rispettare.

Il GDPA è un atto emanato nel 2018 che ha unificato importanti concetti, obblighi e diritti sulla protezione dei dati personali e della privacy. Il GDPR è il regolamento generale per la protezione dei dati personali che tutte le aziende che trattano dati personali dei cittadini UE devono rispettare.

La sicurezza è molto importante, e come i SW, va testata. Esistono molti tool o applicativi che permettono di testare molte sfaccettature della sicurezza di un sistema informatico. Tra queste vi è il fuzz testing (fuzzing), il quale si basa nell’inserimento di stringhe malformate al fine di ottenere comportamenti anomali e trovare falle. Ci sono delle varianti, in base a quanto si conosce del codice sorgente o il bersaglio.

La direttiva NIS, adottata in UE nel 2016 e recepita in Italia col decreto del 2018, stabilisce i requisiti minimi per la sicurezza delle reti e dei sistemi informativi nell’UE. Essa ha portato al fornimento di framework veri e propri per la cybersecurity e data protection. La direttiva è applicata a operatori dei servizi essenziali, di servizi digitali, e non si applica a fornitori di servizi digitali che sono considerati piccoli o alle microaziende.

Spesso gli attacchi sono mirati all’anello debole della catena (supply chain), ne è un esempio l’attacco a Solar Winds. La resilienza deve riguardare tutta la supply chain. I fornitori di servizi cloud della Pubblica Amministrazione (PA), devono soddisfare determinate misure di sicurezza, e se le soddisfano possono aspirare ad ottenere qualificazioni cloud. La resilienza della supply chain si raggiunge con la consapevolezza sia dell’organizzazione che del fornitore, tramite audit (controlli), e contratti veri e propri.

La vulnerability assessment è una valutazione delle vulnerabilità che consiste in un’analisi dettagliata, grazie alla quale è possibile prevenire minacce. Vengono testati server, postazioni di lavoro, firewall.

Il penetration test a differenza della vulnerability assessment, è un attacco vero e proprio allo scopo di migliorare la sicurezza del sistema, e verificare che tipo e quanto attacco riesce a reggere. In genere viene svolto su un applicativo a parte, orientato solo al testing per evitare di fare danni al sistema che è in produzione. Ovviamente è accompagnato da tool che in seguito all’attacco, forniscono dei report sull’andazzo dell’attacco.

**LEZIONE 7**

**AUTENTICAZIONE**

Autenticare vuol dire ri-conoscere, cioè, presuppone di avere già una conoscenza. Ne è un esempio quando si rivede una persona dopo la prima volta, dopo la conoscenza, cerchiamo di identificare lo stesso viso, riconoscere quindi. Tutti i servizi ormai la richiedono. A chi fa comodo? Sia a chi si autentica, il quale tutela il proprio spazio dagli altri, e chi autentica, perché deve farlo per legge, ma anche per gestire al meglio e garantire il servizio in sicurezza.

Si basa nel dimostrare chi si è. Questo perché devo inserire dati sensibili, come il numero di telefono, la carta di credito. È un caso di autenticazione anche quando si vuole riconoscere se un sito è attendibile o fasullo, in questo caso è importante non sottovalutare l’URL, spesso discriminante. L’autenticazione è differente dalla tracciabilità. La base della sicurezza in termini di autenticazione è quando noi vogliamo autenticare un sito (per esempio anche i quotidiani sono https e vogliamo leggere notizie da fonti attendibili e valide), ma anche il sito vuole autenticarci per offrirci il servizio.

Per quanto riguarda le misure per applicare l’autenticazione, si ha la biometria, la conoscenza e il possesso. La biometria a primo impatto sembra essere la migliore perché si basa su caratteristiche univoche che ogni persona ha, come l’iride o l’impronta, ma in realtà è al momento la più debole, basti vedere quanto costa un kit di raccolta impronte. Proprio per questo essa viene sempre accompagnata da una password obbligatoriamente. Ci sono tante caratteristiche che ci rendono unici, il modo di camminare, il timbro di voce, e poi ce ne sono altre meno, le quali sono più facili da alterare.

**INTEGRITA’**

L’obiettivo è quello di mantenere il dato inalterato di nessun bit. Non deve esserci alterazione indebita, illecita, malevola, tipicamente quando il dato transita (per esempio dal web server al browser). Un flusso di dati, un pacchetto, sarà sempre alterabile anche di un solo bit. Partirà sempre da un web server e attraverserà vari nodi. TCP-IP ha un sistema di ricordanza, tramite il checksum, che è una misura di integrità (?). Ma il checksum a sua volta potrebbe anche essere alterabile. Cioè, esso funziona, ma ovviamente se si va ad attaccare la funzionalità del checksum stesso, ecco che scatta la falla.

Infatti, mentre alla nascita della rete e dei primi sistemi informatici e distribuiti, si pensava solo ai requisiti funzionali, cioè che tutto andasse, col tempo si è avuta e compresa la necessità di integrare anche requisiti non funzionali, quali su tutti la sicurezza. In conclusione, l’integrità è possibile da ottenere tramite la firma digitale. In sostanza essa usa una funzione più particolare del checksum, cioè l’hash.

**PRIVATEZZA**

Un grande passo in avanti sulla privatezza si è fatto tramite il GDPR. C’è ancora tanta confusione tra sicurezza e privatezza. La privacy è un diritto alla segretezza (che invece è il rispetto della policy) dei dati sensibili (non la password): religione, orientamento sessuale, hobby, vita privata e altro, quindi è a un livello superiore. È legata col big data e va a braccetto col data protection. Alcune misure per mantenerla sono l’interagire il meno possibile (!), e i consensi alle policy, le quali spesso ci chiedono se vogliamo accettare o rifiutare i cookie installati (essi lo fanno solo per legge). Il consenso alla policy è la misura principale.

\*tool(misura) è uno strumento tecnico-organizzativo che realizza una proprietà, quella di sicurezza in questo caso. Le misure per la privacy sono la policy e poi il proprietario deve accettarla. La temporalità è importante, infatti non tutti i DB che tengono i nostri dati lo fanno per sempre, per esempio l’università lo fa per sempre, i contratti telefonici no. Tradizionalmente in USA il dato sensibile è stato merce di scambio, in UE invece è molto più controllato. Basta vedere il caso chatGPT che è stato chiuso in Italia dal garante della privacy, poiché i dati andavano in USA, la quale appunto tratta diversamente i dati rispetto UE. Ecco, quindi, il problema dell’export dei dati.

Per capire l’importanza della privatezza (anche per la democrazia), basti pensare a cosa succederebbe se tutti conoscessero i nostri dati sensibili. Ad un colloquio di lavoro si potrebbe trovare il datore discriminante per qualsiasi cosa, poiché conoscendo quel tipo di informazioni su una persona, cambia la percezione di essa. In qualche modo la banca quando fa il mutuo, applica degli algoritmi (accingendo dati sensibili...) per vedere se quella persona è un tipo affidabile o meno, e altro. La sfera personale è molto importante. Se un servizio ci dà la password, sta a noi mantenerla segreta.

**LEZIONE 8**

**ANONIMATO**

È una specifica della privacy, verticalizzata sull’identità. A differenza della privacy non sempre è un diritto, poiché in molti casi è giusto che non ci sia al fine di responsabilizzare le persone. Questo è molto discutibile, ma è plausibile in certi casi in cui serve conoscere l’identità. A tutti piacerebbe fare determinate cose senza essere individuato, acquisti o ricerche. Supponendo ci sia, il vantaggio è la protezione del dato.

Un esempio è il laboratorio analisi che registra tutte le tuple, che in realtà dovrebbero essere cancellate, altrimenti necessita un motivo giustificabile per tenerli. Leggere perciò articolo del GDPA (32-33 o 34?). Non ci sono dei principi assoluti per il trattamento dei dati, è molto relativo, e il GDPA non sancisce chi può e chi non può, ma il modo di trattare i dati. Ci sono dei servizi che pretendo solo per me, e me in questo caso rappresenta un’identità, ed è quindi necessario un minimo di autenticazione per garantire il servizio. Per questo è fondamentale la pseudonimizzazione, cioè per esempio al posto di mario rossi, scrivo xyz. Da una tupla anonimizzata non ci si può ricondurre al proprietario, mentre nella pseudonimizzazione si dovrebbe poter de-anonimizzare, tramite ulteriori informazioni.

La navigazione anonima su Web in realtà non è tale, semplicemente non viene salvata la cronologia. In realtà siamo localizzabili tramite IP (che è dinamico perché quelli disponibili con IPV4 sono pochi e per questo non possono essere statici), anche se dipende dalla policy per l’IP dell’ISP (Internet Service Provider), anche se esso per legge deve loggare il traffico e quindi collaborare con la legge nel caso in cui ci sia qualche transazione illegale e si vuole scoprire il soggetto. Ma anche l’IP dinamico è un dato sotto registrazione, quindi non c’è anonimato. Esso è presente solo sulla macchina, ma dal lato server non si è anonimi.

TOR è un SW che permette la navigazione “anonima” su web tramite il routing a cipolla, il che rende molto difficile trovare il soggetto di partenza, poiché ogni livello della cipolla deve partecipare, in quanto il nodo più esterno vede solo l’IP del nodo precedente e così via. Quindi su TOR si ha l’anonimato a livello di routing. Da ciò si può dedurre che l’anonimato può essere a diversi livelli architetturali.

A livello di applicazione, l’anonimato è garantito da server anonimizzatori o proxy. Qui si è anonimi in ciò che si fa, cioè l’identità viene esposta solo al proxy, il quale è appunto un servizio border-line, al limite della legalità e molto oggetto di discussione; la richiesta al server richiesta dall’utente verrà fatta dal proxy. Essa viene spesso confusa con la VPN, ma cambia la policy del servizio, in quanto la VPN riesce a fare entrare in server locali, per esempio quello di DMI, tramite VPN si ottiene un IP locale al server DMI (e cambia anche il protocollo). Il proxy nasce come protettore della privacy del nostro IP, VPN ha un obiettivo diverso, cioè quello di avere un accesso globale, per esempio se si vuole guardare un canale della tv italiana dall’ America. L’anonimato di entrambi è però legato alla policy di essi, perché loro in realtà ci conoscono e a loro non si è anonimi, anche qui vi è l’atto di fede in essi. Per questo le VPN al giorno d’oggi, sono molto centro di discussione delle leggi locali e mondiali.

**2° LIVELLO**

**NON RIPUDIO**

Prevede che la controparte possegga evidenza sul fatto che il mittente ha effettivamente partecipato inviando quel messaggio, o acquistato qualcosa. Un esempio in Italia è la PEC (Posta Elettronica Certificata), che fornisce l’evidenza a tutti i livelli, ciascuno si può tutelare dal ripudio dell’altro tramite questi meccanismi e protocolli.

**RELAZIONI TRA LE CARATTERISTICHE**

È importante che le caratteristiche della sicurezza vengano messe in relazione, da sole non hanno molto senso. È ovvio che l’autenticazione implica il non anonimato. Quindi anonimato implica non autenticazione. Infine, autenticato equivale al non anonimato. Supponendo l’autenticazione funzioni, cioè con tutte le misure che abbiamo visto, essa non implica il non-ripudio perché, se per esempio sono l’unico ad assistere ad un crimine, ma non ho una prova, un’evidenza inequivocabile (tipo una foto), allora è possibile il ripudio in quanto il dubbio rimane; chi ci autentica ha il non-ripudio, ma non implica il non ripudio assoluto per tutti gli altri, serve un ulteriore servizio. Quindi si può dire che il non ripudio è “più forte” rispetto l’autenticazione. Ne è un esempio la PEC. Quindi in generale l’autenticità è una condizione non sufficiente, ma necessaria per il non ripudio. Il viceversa infatti è vero, il non ripudio implica l’autenticazione (sicuramente c’è stata autenticazione se c’è non ripudio). In maniera deduttiva l’anonimato, dato che implica la non autenticazione, implica anche il non non ripudio, cioè il possibile ripudio. Anche perciò il non ripudio, dato che implica l’autenticazione, implica il non anonimato.

Queste relazioni non sono assolute, ma è per esempio possibile avere caratteristiche apparentemente inconciliabili, allo stesso tempo presenti con un enforcing, come vedremo nell’esempio del foglio WATA2, poiché essendo importante il “da che cosa” riferito alla sicurezza, ed essendo queste caratteristiche della sicurezza, anche per esse bisogna specificare il “da che cosa” e in quale fase temporale.

**LEZIONE 9**

**FOGLIO WATA 2**

Lo studente vuole l’anonimato, il docente invece necessita l’autenticazione. Apparentemente sono inconciliabili. Quando si fa un esame ci sono delle regole, che sono a tutti gli effetti delle proprietà di sicurezza: il docente vorrebbe autenticare lo studente per questioni formali del compito e per essere certo che sia proprio lui a fare il suo compito prevenendo imbrogli (mandare qualcun altro a fare l’esame è un attacco al sistema), si necessita appunto un’associazione studente-esame veritiera; mentre lo studente vuole l’anonimato per prevenire votazioni inique da parte del docente, per tutelarsi.

Ogni protocollo è un sistema di sicurezza e affinché esso funzioni, la sicurezza stessa è necessaria, e qui entra molto in gioco il fattore umano. Bisogna che entrambe le parti del protocollo seguano le regole per ottenere e preservare le proprietà di sicurezza, poiché in caso contrario le proprietà si rovinano e vengono meno. Nel caso in cui la controparte del protocollo sia in rete, è difficile verificare che la controparte rispetti le regole; quindi, si ricade sempre in una dose di fiducia.

Lo studente quindi si deve autenticare, ma allo stesso tempo vuole l’anonimato dell’esame perché serve fede sull’imparzialità del docente. Tutti i partecipanti devono seguire le regole del protocollo e ciascuno si deve tutelare da possibili deviazioni che inficerebbero le proprietà di sicurezza. Ma come fanno le due proprietà a coesistere se abbiamo visto che una implica il contrario dell’altra? È importante, come detto nella lezione scorsa, il complemento di causa efficiente, il “da che cosa”. Il docente necessita l’autenticazione del compito e dello studente, e deve anche sorvegliare per evitare che gli studenti si scambino i compiti; necessita quindi di queste 3 cose, se ne manca anche una sola il sistema è violato.

Ma lo studente non è ancora anonimo. È anche un problema visivo perché qualunque protocollo ci sia, il docente può fare associazione volto compito e può quindi non essere imparziale. Quindi bisogna specificare anonimato da che cosa. Quest’ultimo tipo di anonimato sarebbe inusuale, il professore non dovrebbe essere all’esame, e lasciare il compito a un sorvegliante.

Anche l’ordinamento dei compiti è importante, e rimescolare la pila dei compiti è una misura di sicurezza, laddove l’obiettivo sia l’anonimato. In WATA2 è lo studente che risolve l’associazione compito-studente. Il compito viene suddiviso in token (talloncino con anagrafica dello studente) e il compito vero e proprio, abbinabili tramite ID.

Lo studente si porta il token, ma se lo perde, perde anche il compito poiché solo lui può risolvere l’associazione; il token deve essere robusto all’attacco fisico, in primo luogo il docente controlla il documento d’identità e lo confronta col token, ma il problema viene dopo poiché un attacco ovvio sarebbe violare il token, scambiando il codice a barre con un altro con cui lo studente si mette d’accordo, e in difesa di ciò si ha una firma sul token del docente, in particolare tra il codice a barre e l’anagrafica per renderlo meno copiabile possibile; Il vigilante potrebbe essere diverso dal docente ed essi non devono colludere; dato che le domande sono randomizzate, e sicuramente tutti avranno la prima domanda diversa, questa sarebbe un discriminante per ogni studente, quindi al passaggio del vigilante, durante il controllo dei documenti, lo studente copre tutto tranne il token, o piega il foglio; l’ordine dei compiti è importante anche alla consegna; dato che il compito del token è quello di accoppiare in maniera univoca il compito con lo studente, contro la violazione del token, l’aggancio, la firma viene messa a cavallo tra dati e bar-code, in modo da rendere più difficile la copia del bar-code.. quindi la firma serve per certificare che bar-code e dati si riferiscono alla stessa persona; i token sono pre-firmati o anche post-firmati, non cambia, importante è che il docente autentichi lo studente.

Quindi il nocciolo autenticazione-anonimato ha trovato un raccordo, perché autenticazione e anonimato sono relativi rispetto a cosa e quando, infatti l’anonimato termina dopo la correzione.

Il protocollo è suddiviso in 4 fasi: preparation, testing (esame degli studenti), marking, notification.

* Preparation

Randomicamente l’esaminatore estrae una t-upla di domande, stessa cosa per ID del test tramite una stringa di n caratteri, inizializza il DB dei voti con id\_test e blank (qui ci andrà il voto), stampa i singoli test, poi test signed è l’insieme dei test che verranno distribuiti.

* Testing

Qui vi è l’autenticazione dello studente.

Il compito arriverà randomicamente al candidato, per evitare che il docente dia un compito difficile in maniera intenzionale, in un qualsiasi modo, e lo studente compilerà il token. Qui il candidato va autenticato dal vigilante, tramite il documento d’identità, ne controlla la validità dello stesso, poi controlla tutto il registro dei prenotati all’esame, e infine confronta token con documento d’identità. I controlli in realtà sono 4 perché confronta anche la foto con il viso dello studente. Il vigilante restituisce documento e compito. Poi lo studente scrive il form del test, e alla fine taglia il token producendo un a\_test(anonymous). A random lo dà al vigilante, in qualche modo.

* Marking

Qui vi è l’anonimato del compito.

Il docente corregge e assegna i voti, inserendo il voto nel DB con coppia (voto, QR).

* Notification

Lo studente si reca dal docente e avviene l’accoppiamento QR del token studente con QR del DB del docente, il quale registra (id studente, corso, voto) nello storico, sia per evitare conflitti con voti passati e in caso per registrare in un portale studenti. Però anche qua il docente potrebbe dire un voto ­­diverso da quello del sistema se egli vuole. Il protocollo per essere totalmente sicuro dovrebbe infatti essere integrato col portale studenti per evitare attività malevola del docente.

**LEZIONE 10**

**DISPONIBILITA’**

La proprietà di disponibilità (avialability) è il contrario di DoS (Denial of Service). È importante che il servizio sia funzionante per essere sicuro, se esso si confonde non sarà ben funzionante, quindi non safe. Diventa una misura di sicurezza nel momento in cui l’attaccante ha come obiettivo quello di interrompere il servizio. Una misura per garantire la disponibilità è molto difficile perché il DoS può anche essere distribuito. Non bisogna confondere la disponibilità con l’autenticazione. Fare DoS potrebbe significare fare fork-bomb, cioè, fare richieste in loop saturando le risorse del sistema, la difesa potrebbe stare nel far fare un calcolo prima della fork all’attaccante, il quale si saturerà prima della vittima; quindi, lo impegnamo computazionalmente per evitare che le faccia in cascata. Un esempio di difesa, perciò, è il rate-limiting (fornirò un servizio peggiore limitando le richieste, ma evito DoS).

**3° LIVELLO**

**CONTROLLO D’ACCESSO**

Esiste una policy che deve dire chi può fare cosa. Esempi come cosa può fare root, cosa può fare giamp, cosa può fare il docente e cosa lo studente (vari esempi di policy anche nella vita). Delle misure potrebbero essere l’autenticazione dell’utente, poiché se non lo riconosco non posso pensare all’autorizzazione, le policy, e implementazioni di esse.

**ESEMPIO DI POLICY**

Le policy sono un insieme di regole, di qualunque tipo. Scrivere una policy è molto difficile per via delle ambiguità e delle inconsistenze. Le regole stanno definendo il senso delle cose descritte, per esempio un file pubblico è descritto dalla policy stessa. Downgrade = portare alla versione precedente. Anche l’utente segreto stesso è definito dalla policy stessa. A seconda di quali sono le relazioni tra il tipo di utente tutto ciò potrebbe essere inconsistente. Per scrivere bene la policy, e anche per intrepretarla bene, bisogna conoscere il sistema e i vari ruoli che girano lì dentro. Ogni persona impiega un ruolo in tempi e luoghi diversi. Non bisogna schiacciare l’utente al ruolo, sono cose diverse. Per discrezionalità si intende una sorta di libertà di scelta (“a discrezione tua”). In una policy devono anche essere chiare le relazioni tra i ruoli. Tutte le modalità derivano da quella base di ‘obbligatorio’. È bene disambiguare i termini usati. Nella slide vi è una possibile intersezione tra i ruoli, discutibile, ma in questo caso è questa. Facendo la policy decidiamo noi cosa è giusto o sbagliato, poi il resto è una questione etica.

Le inconsistenze sono di due tipi, ridotte alla modalità base dell’obbligatorietà: il dilemma si ha quando vi è un obbligo su una cosa e sulla negazione della cosa, mentre nella contraddizione la negazione sta nella proprietà, l’obbligatorietà in questo caso. Passare a un sistema formale rivelerebbe le inconsistenze perché ci sarebbe tutto da scrivere, ogni sfumatura. Le singole regole sono elementari. Il problema è l’insieme delle regole e le relazioni tra esse che possono facilmente portare inconsistenze. Sono importanti, per esempio, quando si scrivono firewall, basati su policy e nei SO. Anche disgiungendo tutti i ruoli e privare le entità di relazioni, si dovrebbe riscrivere l’intera policy per ogni ruolo, e il problema c’è lo stesso, e soprattutto non è scalabile.

La soluzione di queste inconsistenze è quella di dare delle priorità alle regole, in ordine di come le scriviamo (top-down), ma senza scordarsi regole sotto perché diventerebbe bottom-up. C’è regola di default: denay -> cioè, tutto ciò che non fa matching con regole non è permesso, anzi è una metaregola, cioè una regola delle regole. Si parla in questo caso di modello, cioè il modo di dare le regole, di scrivere le policy.

**MODELLI DI POLICY**

MAC è basato su mandatorie non modificabili, di provenienza e ispirazione militare, vincola molto ed è immutabile. Usato in SELinux e AppArmor. Nei nostri SO comuni invece, per esempio, root può fare tutto, in questi sistemi non è così. Mentre RBAC è il contrario, non mandatorio, su cui si basano i nostri SO, basati su permessi associati a ciascun ruolo.

I meccanismi implementativi spiegano come applicare le policy per garantire la proprietà di controllo d’accesso. ACM ne è un primo esempio, sta per matrice di controllo d’accesso. È una griglia in cui si scrivono soggetti e oggetti, chi può fare cosa su cosa. Ciascuna cella dice i permessi sul dato soggetto su un certo oggetto. Nel caso in cui il permesso, la modalità sia una sola, non si avrà necessità di scriverla, ma basta un tick o cross.

Qualcuno pensò che potrebbe essere più comodo registrare queste matrici, per linea o per colonna; è il caso di ACL, che raccoglie la riga della matrice con i permessi legati allo specifico oggetto (con i seguenti utenti). È identica a quella di Linux, la differenza è che in Linux si ha proprietario, gruppo di appartenenza e other. L’altro approccio è Capability List (CL), cioè, descrivo tutte le proprietà in funzione di un soggetto (con i seguenti oggetti).

**LEZIONE 11**

**TASSONOMIA DI ATTACCANTI E MODELLI**

Differenti tra diritti, esperienza, risorse (per esempio il pen tester ne ha molte ed è molto pericoloso, stessa cosa per il rischio accettato), rischio accettato. Si possono distinguere anche per i moventi: un primo può essere la ricchezza, fare soldi; altri invece specifici sui dati personali e sensibili, o anche potere gloria e divertimento. In basso c’è invece una lista generica di nomi che essi possono prendere. La difesa è per esempio la polizia postale.

Quindi anche qui posso dire che sicurezza dipende da che cosa, da quale tipo di attaccante. Quindi come facciamo a parlare di sicurezza universale? Si applica un dibattito analogo alla complessità asintotica degli algoritmi, in cui si considera il caso peggiore (soprattutto, anche se poi si ha un approccio risk based), medio e migliore. Posso astrarre tutti questi tipi di attaccanti e avere un modello universale? Posso farlo perché basta guardare dall’alto le cose astraendole, salendo di livello di dettaglio, generalizzando di più. Però ci possono essere tanti modelli, quindi se si è sicuri verso un modello di attaccanti, cambiando il modello magari non lo si è più. Qual è però il migliore? È difficile definire il modello di attaccante che scinde bene tutti i tipi di attaccanti, non sono definibili nella loro interezza. Quindi in genere convenzionalmente si mettono una serie di attacchi in un gruppo, forzando la cosa, che è discutibile. Perché ha senso fare un modello di attaccante? Perché basandosi sulla realtà, bisogna definire la sicurezza da che cosa.

Nei problemi di sicurezza distribuiti il modello usato è Dolev-Yao (DY). Essi negli anni ’70 hanno cominciato a porsi questi problemi universali, sulla descrizione degli attaccanti in caso di sistemi distribuiti. Nel caso peggiore si può pensare che le capacità offensive colludano (allearsi, condividere capacità), cioè si ipotizza di mettere tutti gli attaccanti insieme. Si assume che non violi la crittografia.

Col tempo però le capacità degli attaccanti diventavano più ampie e capillari (quindi disponibili per tutti praticamente), quindi nasce il modello General Attacker (GA) che non ha bisogno di colludere, sia per questioni di disponibilità di risorse e conoscenze, sia per fini diversi (poiché nel modello precedente la collusione si aveva perché il fine era uguale tra i colludenti).

Qual è il modello “giusto”? “Giusto” è da contestualizzare. Qual è il modello “più forte”? Non si può dire perché bisognerebbe analizzare ciò per tutti i sistemi target possibili.

In WATA supponendo una sorta di general attacker (senza collusione tra vigilator e docente o che entrambi siano la stessa persona, e dal punto di vista degli studenti ognuno non ha interessi nel voto degli altri), potrei togliere la randomizzazione perché il vigilator non avrebbe interesse nel colludere col docente, e quindi sussiste l’anonimato dello studente. Se invece assumo che i due colludano (una sorta di Dolev-Yao), il protocollo senza la misura di randomizzazione ha un attacco. Anche se sembrano due modelli banali e spesso non utilizzabili, in realtà una domanda da farsi sempre è quella del modello di appartenenza, quando si fa security by design. Quindi si fa una lista delle capacità dell’attaccante e poi si associa al modello più corretto. In entrambi i modelli si assume che la crittografia non sia violabile perché si suppone che quegli attaccanti non ne hanno la capacità, è una scelta, per avere il focus sul resto della sicurezza.

**AUTENTICAZIONE**

Quella più usata è sullo smartphone, esiste anche autenticazione tra macchine tramite IP, o tra utente-utente tramite Kerberos che è un protocollo. Concentriamoci sull’autenticazione computer-utente. Un esempio è quando si inserisce un sistema di pagamento su internet, ma dobbiamo autenticare il sito (riconoscerlo, capire che è qualcosa che noi pensiamo sia) per fidarci di darlo.

L’autenticazione utente-computer si può basare su 3 cose: conoscenza, possesso e biometria. Spesso vengono usate in combinazione, per esempio il bancomat ha autenticazione a due fattori (2FA): possesso e conoscenza. Potrei fare un bancomat con solo biometria? Si, ma sarebbe poco sicuro, perché sono facili da lasciare in giro e tracciabili, e inoltre sono potenzialmente uguali su più sistemi, mentre si può mettere una password diversa per ogni sistema di autenticazione; quindi, la biometria è più usabile che robusta. Il fatto di cedere la carta a qualcuno di cui noi ci fidiamo è un problema che parte dalla violazione della policy, perché nella carta ci sta il nome del proprietario.

**BASATA SU CONOSCENZA**

Slide 76 e 77. Slide 78. L’attacco vocabolario è reso inutile col controllo a soglie. Il test di Turing del CAPTCHA è un modo per testare l’intelligenza della macchina, se questa era capace di simulare un attaccante e non essere distinto. La CAPTCHA è di per sé un test, ma ci sono appunto attacchi al test di turing del CAPTCHA, e per questo se ne sono costruite sempre più difficili. Si evince che il sistema CAPTCHA è arrivato alla sua fine, perché gli algoritmi contro sono sempre più affidabili. Per questo c’è il sistema reCAPTCHA di Google, si basa solo su un click nel cui si dice di essere umani. Approfondire su cosa si basa. Probabilmente per tutto quello che sa Google, non è open-source, e non si sa nulla. Slide 79. Il migliore bilancio tra robustezza e mnemonicità della password è fondamentale, in genere è meglio averla di almeno 8 caratteri.

Le password sono il mezzo principale per l’autenticazione, il ri-conoscimento. Esse però devono essere mantenute da qualche parte, in genere in un DB delle password. Questa è un’idea del 1960 e il sistema si chiama CTSS. Si confronta la password live, che sta in RAM, con quella che sta nella tabella del DB. Questo è il primo sistema multiutente, è stata la prima istanza in cui si necessitava l’autenticazione. Si ha circolarità dell’idea perché a loro volta le password dei DB devono essere protette da un’autenticazione, oggi tramite algoritmi Hash.

**LEZIONE 12**

**BASATA SU POSSESSO**

Da un punto di vista razionale è uguale a quello su conoscenza, perché come si può cedere la conoscenza, si può cedere l’oggetto di autenticazione. In genere sono delle card o token. Non c’è un sistema più debole dell’altro, perché è sempre responsabilità del possessore. La migliore cosa è combinare una password con un oggetto, costituendo un’autenticazione a due fattori.

Gli smart token hanno avuto un ciclo di vita molto breve, poiché oggi sono diventati SW. Essi sono dei “potenziamenti” rispetto agli oggetti classici. Prima era necessaria un’autenticazione basata su conoscenza allo smart token. È la più evoluta autenticazione basata su possesso. Il server dà una password momentanea (OTP), one time password, che l’utente può leggere solo dallo smart token; quindi, è implicito che egli lo possegga. È una password non statica, ma dinamica. Lo scopo è comunque sempre quello di dimostrare al server che si possiede lo smart token. Ciò porta complicazioni nell’attaccante non di poco conto, perché la frequenza di cambio irrobustisce la difesa. Ciò deve però implicare il fatto che queste OTP passano per protocolli di sicurezza, per esempio su internet tramite https. Quindi misure di sicurezza ce n’è sono più di una. Adesso tutto è spostato su app, quindi è un sistema traslato, ma lo usiamo più spesso di quanto crediamo. È un passaggio da un sistema chiuso a uno aperto, è un ulteriore atto di fede allo smartphone che deve essere sicuro, poiché esso è collegato ad internet e quindi è accessibile potenzialmente dal mondo.

Che differenza c’è tra autenticazione bancomat e sito web? Prima il bancomat era protetto da una password, e dal possesso della carta, perché la solo password sarebbe stata troppo debole, tramite un banale keylogger si potevano intercettare le password. Per questo ci si è evoluti con i due fattori negli smart-token.

Che differenza c’è tra il pennino con un pulsante e la carta con la tastiera? La carta è più sicura perché il pin viene messo su carta, non poi su web, perché lì aumentano le vulnerabilità; invece, tramite solo la carta si usa in sistema stand-alone, isolato dal resto del mondo.

Perché serve un OTP? Il pin del nostro bancomat non cambia, perché non è one-time. Essa è più forte del pin del bancomat seppur basata anche su conoscenza, poiché one-time. Si vuole quindi la garanzia che quell’oggetto è posseduto da quella specifica persona in quel preciso istante. Le smart-card sono più difficili da clonare perché è molto difficile conoscerle. Ad oggi a causa dell’evoluzione della tecnologia sono confondibili il possesso e la conoscenza.

**BASATA SU BIOMETRIA**

Si dice che abbiamo caratteristiche uniche, fisiche e comportamentali. La rappresentazione della password è più accurata perché digitalmente si conoscono e sono univoche per ogni lettera e quindi ogni combinazione, mentre riguardo il convertire un’impronta, un’iride o il timbro della voce dovrebbero essere convertiti in digitale e non si può essere precisi e accurati al 100%. Quindi non si può prescindere da questo campionamento. L’impronta live sarà inoltre sicuramente diversa da quella del template, quella salvata, generando molti falsi negativi a differenza della bontà della password. Il problema di campionamento si ha quindi sia in fase di registrazione del template, sia in fase di acquisizione live per accedere.

Storicamente le impronte erano prese con l’inchiostro, poi con l’avanzare della tecnologia ci si è dovuti inventare lo scanner. Lo schema di tipo loop è più comune rispetto Arch e Whorl. Le tecnologie mirano molto nel registrare le minuzie, cioè punti di inizio e fine di una biforcazione.

**CENNI DI CRITTOGRAFIA**

È un qualcosa di antico, dai tempi dei Greci. È la scienza che tratta la trasformazione di un testo in chiaro in uno non comprensibile. Criptare è come cambiare lingua, tradurre in entrambi i versi, encryption e decryption. Nel cambio di lingua, come nella crittografia, cambia la sintassi, ma si mantiene la semantica. È differente dall’hashing poiché in quest’ultimo caso non si può fare l’azione reversibile. È costituito da input, key (parametro di input della funzione di conversione) e output in entrambe le operazioni. L’unica cosa che viene mantenuta privata è la chiave, l’algoritmo si conosce.

Ci sono due varianti, quella asimmetrica è molto più recente. ENIGMA è una macchina tedesca che cifrava le operazioni, in guerra, dove si dovevano nascondere informazioni ai nemici. Oggi si usa molto in protocolli di sicurezza come https. Un crittosistema è costituito da una coppia di algoritmi, di encryption e decryption, che prendono in input un testo e una chiave. Nel caso della decryption l’input è cifrato e l’output decifrato e viceversa per la encryption. Non necessariamente la chiave è uguale per entrambe le operazioni. Nel caso di crittografia simmetrica la chiave è uguale in entrambi le operazioni, mentre nella asimmetrica sono una l’inverso dell’altra. Nel cifrario di Cesare la chiave è uguale, quindi è un crittosistema simmetrico. Esso è oggi un sistema molto debole.

**LEZIONE 13**

Tutto si gioca sulla segretezza della chiave, ma i sistemi crittografici sono pubblici, cioè l’algoritmo ed eventualmente la lunghezza della chiave. Per esempio, il TPM della nostra macchina contiene delle chiavi. Se l’attaccante scoprisse la chiave tutto il sistema salterebbe. L’attaccante ha sempre una probabilità non nulla di scoprire il segreto, talvolta non trascurabile, poiché si può sempre provare a indovinare. Se la chiave fosse lunga un bit, avremmo il 50%, due bit il 25%, quindi questa probabilità diminuisce all’aumentare della lunghezza della chiave, in particolare è inversa all’esponenziale con base 2 e lunghezza della chiave come esponente. Ma questa probabilità dipende dal fatto che si conosca la lunghezza della chiave, se non si conosce allora sarà più difficile. È importante che poi la probabilità concreta non sia più alta di quella minima (magari con conoscenze ulteriori che aumentino significativamente la probabilità), in quel caso sarebbe un sistema debole, al contrario è sicuro.

Che differenza c’è tra password e chiave? Ambedue devono rimanere segreti per chi li deve condividere, entrambi rappresentati in bit, ma tipicamente una chiave è più lunga, e inoltre la chiave viene usata per cifrare.

**CRITTOGRAFIA SIMMETRICA E LIMITI**

Nella crittografia simmetrica, a chiave condivisa (o anche segreta, perché il segreto deve essere condiviso per essere utile), la prima è proprietà di correttezza, la seconda di robustezza perché dice che tutte le volte che usiamo la chiave sbagliata, il messaggio sarà diverso. Qui c’è una share key, chiave condivisa. L’attaccante può intercettare il messaggio, ma non può capirlo perché cifrato, a meno che non conosca la chiave. La velocità della crittografia è strettamente legata alla lunghezza delle chiavi.

Dato un agente A, la chiave sua sarà “ka”. Bob ha una sua chiave “kb”, ognuno ha la sua. È valida a lungo termine, dopo di che scade, e questo serve perché più il tempo passa maggiori saranno le probabilità di scoprirla. Ma aumentando la lunghezza, la probabilità schizza esponenzialmente, e quindi perché per esempio Office la cambia ogni 6 mesi? Ci sono anche le chiavi e breve termine, o chiavi di sessione. Il problema è come fa Alice a dire a Bob la chiave a lungo termine? Non avrebbe senso che A condivida la chiave con il sistema B, perché ognuno vuole mantenere i suoi segreti e condividerne solo alcuni con determinati agenti. Tramite una chiave di sessione o a breve termine condivisa tra Alice e Bob. Dalla chiave a lungo termine si negozia quella a breve. Ma se i sistemi a comunicare sono più di due? Serve una chiave per ogni coppia di agenti, e ciò è dispendioso. Tramite il protocollo Diffie-Hellman si risolvono questi problemi, o anche tramite crittografia asimmetrica.

**CRITTOGRAFIA ASIMMETRICA E LIMITI**

Si chiama anche a chiave pubblica. Qui esiste una coppia di chiavi, cioè per esempio RSA1024 è un crittosistema con encryption, decryption e un input di 1024 caratteri, genera una coppia di chiavi contestualmente. Alice qui ha una coppia di chiavi a lungo termine. Il punto è che se ti do una chiave non puoi prendere l’altra, una è pubblica e l’altra è segreta. Il primo è l’enunciato di correttezza, se uso la chiave inversa riesco a decriptare il messaggio. Il secondo è enunciato di robustezza, se uso una chiave che non è l’inversa di quella di partenza non otterrò il messaggio decriptato (ne ottengo uno errato). In generale è più lento di quella simmetrica per motivi tecnici e anche per la lunghezza delle chiavi.

Avendo qui una coppia di chiavi per agente, progettualmente si hanno 4 scelte diverse di chiavi per il nostro sistema crittografico. Scenario 1: Alice manda m cifrato con kb-1, si otterrebbe segretezza, ma ricordare che Alice non può avere kb-1 perché segreta di Bob, quindi scenario impossibile; scenario 2: Alice manda m cifrato con ka-1, quindi si potrebbe decifrare con ka, pubblica (!), otterremmo autenticazione del mittente, Alice, non segretezza, in questo caso però oltre a conoscere la chiave pubblica, chi deve decriptare è fondamentale che conosca l’appartenenza della chiave pubblica (Alice in questo caso); scenario 3: Alice invia m criptato con ka, ma si può decriptare solo con ka-1 che non conosce nessuno , solo lei, quindi non si ottiene nessuna proprietà; scenario 4: Alice manda m cifrato con kb, si ottiene segretezza perché solo B può decifrare con kb-1 che ha solo lui, attenzione che anche qui si suppone che Alice debba avere con certezza la chiave di Bob, kb, altrimenti B non potrebbe decifrare mentre un altro in particolare sì.

La slide realizza solo l’autenticazione di Alice a Bob supponendo quindi che tutti potessero decifrare il messaggio tramite la chiave pubblica. Alice vuole inviare il messaggio a Bob, criptandolo tramite una chiave privata (ka-1) (è una convenzione). Quella privata viene salvaguardata. Questo protocollo in particolare permette cifratura in chiave privata e il contrario con pubblica, si poteva anche fare diversamente.

Ovviamente l’algoritmo protegge l’input oltre che il messaggio. Se il sistema crittografico fosse perfetto, cioè se potessi decriptare il testo correttamente solo con la giusta chiave, ci sarebbero ancora limiti (Attaccante ha sempre possibilità di indovinare(?)). Eve è triste perché non può rompere la proprietà di autenticazione perché non conosce la chiave privata di Alice e non la può impersonare.

In conclusione, in base a come si cifra si ottiene una certa proprietà. La chiave privata vuole essere un segreto a lungo termine. Ricordare che A e B hanno una coppia ciascuno di chiavi diverse. Quindi possiamo supporre che combinando le due cose otteniamo entrambe le proprietà, è importante l’ordine. Prima cifro con kb, poi con ka-1, così ho segretezza perché potrà essere decifrato solo con kb, e anche autenticazione di A perché solo lui può cifrare con ka-1. Decifrando si devono invertire le operazioni, al primo passaggio non otterrò nulla di comprensibile, serve la seconda decifratura; quindi, ancora non ottengo nemmeno autenticazione di A, perché non capisco, serve decifrare con kb-1, e poi so di aver ottenuto segretezza e autenticità di A. Però facendo così tutti potrebbero innanzitutto decifrare il primo livello, e un eventuale attaccante inserire elementi. Ma questo è un attacco? No, perché non va contro la policy del sistema crittografico in questione, più che altro un attacco di negazione del servizio. Invertendo invece le operazioni già al primo passaggio si otterrebbe qualcosa di intellegibile, cioè prima cifrando con ka-1 e poi con kb. Qui, infatti, un attaccante non può decifrare nemmeno il primo livello, solo B può. Quindi scelta più robusta. In entrambi avrò conferma del funzionamento alla fine dei due passaggi.

Nell’ autenticazione non importa il messaggio, ma solo la “firma”, come quando si risponde al citofono. È importante l’autenticazione della chiave pubblica, è un’assunzione fondamentale della crittografia asimmetrica, si chiama certificazione, ed è verificabile da tutti.

Guessing sta per indovinare qualcosa. Un crittosistema è sicuro quando l’attaccante fa decryption e non facendo guessing della chiave, cioè, ottenendo n! = m, la sua probabilità di successo non aumenti tramite la conoscenza di n (chiave errata). Ecco che il cifrario di Cesare non è un crittosistema sicuro, poiché variando le chiavi inserite ed eliminando quelle sbagliate, le probabilità aumenteranno.

**LEZIONE 14**

**FUNZIONI HASH**

Un hash crittografico è una funzione con un dominio ben definito, e la complessità è bassa, tutti lo possono fare. Fare il contrario è molto difficile invece, ovviamente l’obiettivo dell’hash è quello di renderlo impossibile, però basti pensare a MD5 che è stato rotto. Modificando il messaggio m in m primo, l’hash di m primo sarà diverso dall’hash di m.

Piccole variazioni dell’input portano grandi variazioni dell’hash di output. Il primo utilizzo dell’hash si ebbe col CTSS, tramite il quale il file delle password è hashato, in modo che chiunque riesca a violare e accedere a esso, vedrebbe comunque nulla in chiaro. Infatti, quando si fa un confronto di password, esso avviene nella sua versione hash, cioè prima si converte l’input in hash, e poi si confronta con la tabella della password sempre in hash.

Il problema delle password deboli si è percepito presto. Il sale (bit) aggiunto all’input serve per rendere più sicura la password, sale sta per rumore, entropia. L’hash viene applicato alla password col sale. L’approccio Unix per l’implementazione è descritto in slide, storicamente. In particolare prende una funzione crittografica crypt(3) (molto famosa), basata su DES. In questo modo descritto funziona molto come una funzione hash. Si usa la password per fare l’encryption perché è l’unica cosa che ho, sennò l’utente dovrebbe inserire anche una ipotetica chiave. Quindi in questo modo si ha segretezza della password, che è cifrata tramite una stringa di zeri, si ottengono le stesse proprietà dell’hash.

Dalla slide dell’algoritmo di aggiunta password, si nota come se una password usata per cifrare ha più di 8 caratteri, non cambia nulla dal punto di vista della sicurezza. Oggigiorno il sale ha più di 12 bit, c’è una certa funzione detta pepe. Salt viene registrato in chiaro.

**FRESHNESS**

La freshness è un attributo della segretezza e dell’autenticazione, perché esse hanno validità se vengono fatte spesso, non una sola volta all’inizio. Rinfrescare i segreti è importante per renderli più sicuri, perché più tempo passa più si indeboliscono ed è probabile che vengano scoperti. Una misura intuitiva è la timestamp. Un’altra è la nonce, è un numero random usato solo una volta. Il problema dei timestamp è che hanno senso solo se si hanno orologi sincronizzati. Esiste il protocollo NTP che permette la sincronizzazione tra l’orario del nostro pc e dei server legati a orologi atomici. Gli algoritmi di timestamp sono più recenti, anche se sono nati prima, perché appunto avevano problemi tecnologici di sincronizzazione. Cambio periodico dei segreti per renderli robusti (segretezza), autenticazione va fatta sul momento. Il timestamp viene dato da chi vuole dare freshness, la nonce viene data da chi vuole ricevere freshness.

Nonce funziona con un numero random (non guessable) generato ad un certo orario, e passato agli interlocutori in modo sicuro. Tutti i messaggi che riceverò che citino quel numero allora garantiscono freshness poiché prima dell’orario di generazione del numero randomico è impossibile che ci siano numeri uguali. Tutto ciò supposto che il numero non sia guessable. È importante sottolineare che è usata una sola volta.

SCENARIO 1. A->B: Na, B->A (m,Na). A vuole freshness, se un attaccante Eve intercettasse il numero e lo modificasse, a B arriverebbe il numero modificato e invierebbe il messaggio con il numero modificato. Questo non è un attacco alla freshness perché Eve (Dolev-Yao) modifica il numero, e A semplicemente non considera il messaggio. Se invece modificasse il messaggio (con uno eventualmente vecchio) mentre il numero random no, darebbe a bere ad A il suo messaggio, facendole credere che è freshness, e questo è un attacco. La nonce deve quindi essere protetta.

SCENARIO 2. A->B {Na}kb, B->A (Na,m). Idealmente potremmo usare la nonce cifrata con la chiave pubblica di B, in modo da garantire segretezza della nonce. Solo B potrà decifrare con la sua chiave privata e avrà la nonce giusta. Ma ancora una volta Eve potrebbe intercettare l’altro messaggio, con la nonce giusta, quindi può ancora attaccare.

SCENARIO 3. A->B: Na, B->A {m,Na}kb-1. A manda in chiaro la nonce, B manda il messaggio con la nonce criptati con chiave privata di B. Eve anche intercettando la nonce, non può creare messaggi ad hoc né modificare il messaggio inviato da B, perché A si aspetta di decodificare con chiave pubblica di B, e questo messaggio si sa per certo che è inviato da B perché solo lui ha la chiave privata per criptare. Non c’è segretezza del messaggio, ma autenticazione di Bob ad Alice con freshness.

Concentrarsi sull’obiettivo della nonce, cioè la freshness del messaggio aldilà della confidenzialità e dell’autenticazione o integrità del messaggio. Quindi vedere per ogni scenario quando è possibile che l’attaccante la violi, cioè che A ritenga dei messaggi recenti quando invece sono messaggi non recenti. La nonce può quindi essere inviata in chiaro quando è una challenge per la freshness.

**LEZIONE 15**

**PROTOCOLLI DI SICUREZZA BASILARI**

Nella nonce Alice ha il suo “orologio”, non gliene frega che il suo orologio non sia sincronizzato con gli altri. Con i timestamp esso viene inserito da chi vuole dare la garanzia di freshness, cioè Bob. Avremo assunzioni massimamente stressanti: crittosistema sicuro e DY. C’è ambiguità notazionale tra chiave a lungo termine in simmetrica e chiave pubblica in asimmetrica. Tutti i messaggi li abbiamo chiamati atomici. La composizione di messaggi è un qualcosa di diverso, si rappresenta con la virgola. La lunghezza della concatenazione non è fissata, ognuno può essere concatenazione di altro. La concatenazione non è un’operazione crittografica ma qualcosa che chiunque può fare (non serve autorevolezza per farla). L’attaccante può modificare sempre anche solo una parte del messaggio concatenato, e quindi non si può autenticare un messaggio concatenato. Alchè con la cifratura si potrebbero autenticare le sue componenti.

L’etichetta a sinistra di un protocollo non ci garantisce che il mittente sia quello, serve una firma(?).

Come si ottiene la proprietà di segretezza tramite un protocollo? Tramite crittografia simmetrica (chiave condivisa), il mittente deve cifrare il messaggio con la chiave condivisa col ricevente (a breve termine); in quella asimmetrica invece il mittente deve inviare il messaggio cifrato con la chiave pubblica del ricevente. Questi protocolli hanno però senso con dei prerequisiti. Nel caso della simmetrica, Alice deve cifrare con la chiave condivisa solo fra lei e Bob, e nessun altro deve conoscerla. Nel caso della asimmetrica, la prima assunzione serve perché, se B ha perso la chiave, qualcun altro potrebbe ricevere il messaggio al posto di B, la seconda perché serve certificazione della chiave posseduta da B. Servono entrambe le assunzioni.

Come si ottiene la proprietà di autenticazione? Tramite crittografia simmetrica il protocollo è identico a prima, ma cambiano le assunzioni. Il prerequisito 2 è fondamentale perché Bob ha certezza di stare condividendo la chiave con Alice, la autentica. Si ottiene anche segretezza perché in precedenza la segretezza ha un solo prerequisito incluso qui. Nel caso asimmetrico si ha un secondo prerequisito importante, cioè che il ricevente possa verificare che la chiave pubblica sia effettivamente del mittente (tramite certificato).

Come si ottengono entrambe contemporaneamente? “Sommo” le assunzioni e le chiavi usate. Nella crittografia simmetrica notiamo come sia tutto uguale a quella per ottenere l’autenticazione. Nel caso asimmetrico si sommano le assunzioni e le chiavi del caso della segretezza e dell’autenticazione.

Come si ottiene l’integrità del messaggio, di cui non abbiamo ancora parlato? 1. A->B: m, h(m). Nonostante si possa sostituire l’hash al posto del checksum, ancora non ottiene integrità, perché l’attaccante potrebbe modificare messaggio e modificare anche hash. Nel caso simmetrico si deve avere m, h (m, k ab) <- si chiama MAC (Message Authentication Code). Nel caso asimmetrico si cifra l’hash con ka-1(Firma digitale, DS = Digital Sign). L’attaccante riesce sempre a modificare il messaggio, però non riesce a fare arrivare al destinatario un messaggio modificato spacciandolo per uno buono, perché poi non può cifrarlo con la privata di A o fare l’hash con la condivisa AB.

**FIRMA DIGITALE**

Gli algoritmi sulla firma sono due: fare la firma, e verificare la firma. La firma digitale eredita i limiti della crittografia asimmetrica, cioè la necessità di certificazione. Notiamo come la firma digitale, a differenza di quella normale, oltre l’autenticazione del firmatario, ha l’integrità in più del messaggio. In slide vediamo come B verifica la firma. Manca un primo passo in cui divide il messaggio dal messaggio hashato. C’è garanzia di integrità sia di una parte che dell’altra del messaggio (messaggio e hash).

Per ottenere tutte e 3 le proprietà, ancora una volta, sommo tutti i prerequisiti e chiavi, hash usati. In tutti i protocolli però, non c’è una misura di freshness.

**INTRODUZIONE AI CERTIFICATI**

Attenzione ai siti con certificati (servono per l’autenticazione), ma scaduti. Come comporre un certificato? Pensiamo a un documento d’identità, esso è composto da dati, una foto, un timbro. Quest’ultimo è fondamentale, perché sennò chiunque potrebbe farlo. Quindi si potrebbe semplificare un certificato in questo modo: la chiave pubblica di A corrisponde ai suoi dati, il timbro che deve essere impossibile da riprodurre o rimuovere, lo replichiamo con la firma digitale, garantendo integrità, e autenticità di chi ha fatto il certificato. Quindi si ha (A, K\_a), h(A,k\_a) K{CA}-1 o semplificando (A,K\_a) K(CA)-1 cioè firmato digitalmente. (ricordare che la firma digitale si fa con la privata). La concatenazione di A,ka, kca-1 (firma digitale) non va bene, perché non è possibile garantire l’autenticità della concatenazione di più messaggi. Il certificato non ha obiettivo di segretezza. Qua si sta certificando l’identità che possiede la chiave privata associata alla chiave pubblica. Però il certificato perderebbe di sicurezza se si indebolisse la firma digitale, cioè una vulnerabilità alla chiave privata usata.

**LEZIONE 16**

**DIFFIE-HELLMAN (DH)**

Ci sono dietro delle assunzioni che si fanno alla magistrale su alfa e beta (sono coprimi), che tutti conoscono. Esso ha l’obiettivo di garantire un segreto tra i due interlocutori.

A genera il proprio Xa, B genera il proprio Xb (entrambi segreti), mentre alfa e beta sono conosciuti da tutti. Algoritmo su slide. Si hanno negli esponenti i segreti di A e B (Xa e Xb). Quindi vedendo in rete Ya e Yb, potrei applicare logaritmo e arrivare ai segreti di Xa e Xb? Log base alpha argomento Ya o Yb. Ma c’è il mod, quindi dovremo avere un logaritmo discreto, ma è calcolabile in tempo esponenziale, intrattabile, quindi l’attaccante non può arrivare a Xa e Xb. Con operazioni locali, e poi scambiandosi Ya e Yb, ciascuno ha calcolato qualcosa che è uguale a quello che ha calcolato l’altro. Per le proprietà delle potenze, sono uguali, e Alice e Bob condividono un segreto. Quindi la robustezza delle chiavi sta nella complessità esponenziale del calcolo. Non si ottiene autenticazione. Quindi è un sistema crittografico simmetrico, e (Yb^Xa) mod b e (Ya^Xb) mod b sono chiavi di sessione e sono uguali.

In conclusione, localmente entrambi calcolano qualcosa di uguale all’altro, dato che il doppio esponente si trasforma in prodotto degli esponenti. È un protocollo corretto (nel senso di correttezza), ma è robusto, cioè, mantiene la segretezza? L’attaccante potrebbe rompere lo schema, cioè, calcolarsi una delle due chiavi, solo con una complessità esponenziale. Gli basta una sola delle due chiavi, perché Ya o Yb sono già calcolati con una delle due chiavi, esse sono già comprese diciamo.

DY “conosce” Xa o Xb implica che DY “rompe” prot. DH, è una condizione sufficiente, ma non necessaria, quindi l’implicazione contraria, cioè che, se DY “rompe” DH allora sicuramente avrà conosciuto Xa o Xb, non deve essere data per scontata, perché serve una dimostrazione che non ci siano altri modi per romperlo. Quindi si era dato per scontato che ci fosse questa implicazione, ma per negarla basta un controesempio, cioè un caso in cui è stato rotto DH senza conoscere Xa o Xb. Si era fatto un atto di fede secondo cui non conoscendo Xa o Xb, l’attaccante non avrebbe potuto rompere DH. Quindi formalizzare bene un protocollo è diventato fondamentale.

E per quanto riguarda l’autenticazione? Non c’è, e quindi se A condivide un segreto con B, ma non c’è autenticazione; quindi, potrebbe averlo scambiato con un attaccante, il segreto non è più tale. Quindi la segretezza senza autenticazione lascia il tempo che trova. Anche per questo ci sono altri modi per rompere DH. Esempio di man in the middle in slide, con attaccante che intercetta Ya, e lo rimpiazza con Yc proprio. A e B avranno due numeri diversi, quindi A e B non staranno più condividendo qualcosa, anche se credono di sì perché si fidano del protocollo e non vedono irregolarità (nel caso in cui l’attaccante non cambi la Y ma il messaggio). Se l’attaccante cambia invece la Y, A e B avranno due segreti diversi. In base all’attacco si ha una duplice violazione all’autenticazione o alla segretezza. Si hanno due fix equivalenti perché?

Serve quindi una misura di autenticazione e sono possibili due tipi di fix, uno all’inizio e uno alla fine. I problemi sono due: Alice pensa di condividere la chiave con Bob; l’attaccante conosce ambedue le chiavi. Se risolvo uno dei due risolvo l’attacco. Tramite DH, simmetrico, non è possibile, ma bisogna ‘potenziarlo’ e averlo asimmetrico. L’encryption simmetrica è molto più veloce.

* Se A cifrasse Ya con la sua chiave privata (siamo nella asimmetrica), B sarà certo che il messaggio proviene da A, anche se C intercettasse il messaggio e mettesse la sua chiave privata, B non potrebbe decifrare con quella pubblica di A. Stessa cosa verrà poi fatta da B.
* Per risolvere il secondo problema, la segretezza, A cifra Ya con la chiave pubblica di B, così che solo B può decifrare. E viceversa B cifrerà con la pubblica di A. C quindi non conoscerà mai Ya e Yb, non potendo costruire Yc. Potrebbe costruire Yc, ma non ci farebbe nulla, perché cifrandola con la sua pubblica, la controparte non capirebbe nulla dopo decrypt.

**RSA KEY EXCHANGE**

Il mittente invia una chiave di sessione inventata randomicamente al destinatario, cifrata con la chiave pubblica di B, così che solo egli potrà decifrare. Però così non c’è ancora autenticazione, quindi ancora non è equipollente con DH fixato. L’attaccante non può trovare questa chiave di sessione, ma può spacciarla per sua nel senso che spaccia la sua kc per kb. Ciò significa che non ha senso senza certificazione. È importante che il certificato sia firmato digitalmente da un’autorità di certificazione (CA) (andare a rivedere paragrafo sulla certificazione). Le autorità di certificazione praticamente vendono trust, cioè ci si fida che i certificati siano affidabili. È importante quanto l’autorità è degna di trust in base al contesto, tradizionalmente i più ricchi, le banche, Google. Quante autorità di certificazione esistono? Verisign, Google ecc…

**CERTIFICATO X.509 e PKI (PUBLIC-KEY-INFRASTRUCTURE)**

Il formato standard è il X.509. C’è il CA, il periodo di validità, informazioni pubbliche del soggetto. In base alle versioni vengono aggiunte altre cose, come nella 2 delle chiavi univoche per il soggetto (vedere nelle slide). La firma sta alla fine, ovviamente cifrata. La prima cosa che bisogna fare con un certificato in mano è verificare da chi è stato fatto, il CA praticamente, e poi applicare l’algoritmo di verifica al certificato. Ricorsivamente una firma deve essere certificata da chi sta sopra nella catena, fino all’apice (CA(0), di livello 0). {CA(0),..,ka(a(0))}kca-1(0), cioè autofirmata, si chiama anche RCA (Root Certification Authority). È l’unico certificato che ha come priorità il fatto che la chiave di firma e chiave firmata sono allo stesso livello architetturale. In genere invece si ha {..CA(n)..,kca(n)}kca-1(n-1), cioè il nostro certificato è certificato da un ente più alto nella catena.

**LEZIONE 17**

**AUTENTICAZIONE DI UN SITO**

La catena deve per forza di cosa terminare su una base di fiducia. Affinché la chiave privata abbia senso serve un certificato dell’appartenenza della chiave pubblica. Si ha questo soprattutto nei siti web. Ciò serve perché per esempio in HTTP non c’è encryption. Vogliamo sicurezza del sito, cioè quanto meno autenticazione (noi dobbiamo riconoscere il sito) e segretezza (spesso mettiamo informazioni importanti). In base a cosa l’utente autentica il sito? Bisogna controllare che l’URL corrisponda alla pagina web renderizzata, ciò dovrebbe essere garantito dal browser, ma esso si fida sul DNS che ci sta dietro. Se la corrispondenza non ci fosse, potrebbe appunto esserci un problema del DNS. Se c’è HTTPS, il protocollo ci garantisce questa corrispondenza tra URL e renderizzazione, necessaria perché appunto DNS non è sicuro.

Quindi ci dà sicurezza a livello applicativo. Qui il certificato viene presentato dal server-web al nostro browser, il quale controlla per il client la catena di fiducia. Quindi qui il gioco della certificazione ha tre parti. Ogni tipo di browser è diverso nelle sue scelte di sicurezza, poiché sono i browser a fare la verifica dei certificati e in particolare l’autocertificato della root della chain of trust. Come root ce n’è sono circa una centinaia abbastanza riconosciuti, e il browser ha uno store di certificati di root con cui poi tenta di fare matching per ogni sito; quindi, ogni browser ne può avere uno diverso dall’altro. Di conseguenza ogni sito ha più certificati, con l’obiettivo di raggiungere più browser differenti e quindi più utenti. È molto importante lo store dei certificati del browser, perché è fonte di root; quindi, serve molta sicurezza a questo DB, perché un attaccante vorrebbe modificarlo e aggiungere un certificato che garantisca al proprio sito di raggiungere gli utenti. Ad oggi è stato fortificato il sistema per aggiungere certificati al browser, l’utente deve autenticarsi.

Generazione malevola di un certificato da parte dell’attaccante:

1. Genera ka e ka-1;
2. Genera kRCAa e kRCAa-1;
3. Costruisce {a,ka}kRCAa-1;
4. Costruisce {RCAa, kRCAa} kRCAa-1;
5. Inserisce certificato root nello store del browser della vittima;
6. Utente accede ad a.com e ottiene lucchetto chiuso, quindi si fida e l’attacco è avvenuto con successo: il browser da ora si fiderà di tutti i siti basati su quella PKI e in particolare quella RCA. L’attacco malevolo sta nel passo 5.

Ad oggi è parzialmente nelle mani dell’utente l’importazione di certificati nello store dei browser, cioè egli può aggiornare quello “assicurato” dal browser. Anche togliere certificati dallo store dei browser è un attacco, si fa un danno a tutti i sottodomini che si basavano su quel certificato.

I certificati possono essere Domain validation o Extended validation. I secondi si basano sulla sicurezza del protocollo per conferire i certificati (la procedura è più invasiva) perché, se esso fosse debole allora sarebbe tutto invalidato. Il primo invece è un certificato di poca validità perché ricevuto tramite un protocollo non molto sicuro.

Cosa può andare storto in una catena di fiducia? In termini di segretezza si potrebbe solo perdere la chiave per la firma digitale delle certificazioni, ma ciò non si propaga al di sotto della gerarchia; la perdita di trust invece, per esempio se si perde la chiave e in particolare qualcuno la ruba, si propaga invece verso i livelli inferiori.

Il browser deve quindi verificare un numero di firme digitali pari al numero di nodi nella catena di fiducia.

**CERTIFICATI SELF-ISSUED**

Ci possono essere tanti motivi per cui si ha un avviso del genere: il browser non trova root valide per risolvere il certificato del sito; il certificato viene validato ma è scaduto o ha una avuto una revoca. Questo capita spesso perché i siti che non hanno certificato, possono generarne uno proprio (la RCA), e appare questa schermata di auto-certificazione, sono però certificati gratis quindi DV, scarsi. La scelta più sicura è la terza, uscire senza accedere. Ma è meglio lasciare l’accesso permanente o temporaneo per una sessione? In realtà l’accesso temporaneo è un’occasione in più per l’attaccante di fare danno. Se togliessimo questa opzione l’utente sarebbe più invogliato a non accedere e a scegliere l’opzione sicura, perché la prima è più temuta, l’accesso permanente. Attenzione che si parla di certificato foglia, del sito web, non root. Inoltre, la scelta permanente è rischiosa dipendentemente su che rete sto, quanta probabilità c’è che ci sia un attaccante, circoscrivendo una rete ristretta. Il browser proprio non riesce a risolvere il problema e scarica il barile all’utente, è un limite.

La Certificate Pinning esiste per fissare i certificati di certi siti, che non devono cambiare, per evitare che un attaccante faccia attacchi tramite un certificato valido, cioè si sceglie quale tra i certificati validi accettare (tra quelli visti o conosciuti); il pinning veniva fatto dall’azienda stessa sul sito. Esso è stato superato da Certificate Transparency, una lista di certificati pinnati da Google.

Ad oggi c’è molto business sui certificati, soprattutto grazie let’s encrypt che dà certificati scarsi gratis.

**CERTIFICATE REVOCATION LIST (CRL)**

A un certo punto un certificato deve essere revocato. Se il proprietario perde la chiave privata lui stesso revoca. La CRL è firmata dalla CA che ha emesso il certificato revocato. Di per sé una CRL è un crittotesto, un certificato di revoca certificato. Revocare infatti presuppone un’autorità pari a certificare, perché sennò potrei revocare i certificati degli altri. Il problema è che la CRL deve raggiungere l’end-point, cioè tutti i browser del mondo. Una soluzione non scalabile è che l’utente deve ricordarsi di scaricare anche le CRL presso le CA dei certificati che stava usando, è deprecato l’uso statico delle CRL. Si è risolto tramite il protocollo OCSP (Online Certificate Status Protocol), esso controlla dinamicamente, presso la CA, lo stato di validità del certificato relativo all’URL che l’utente voleva visualizzare. Cercare come il nostro browser verifica le revoche. L’utente non ne sa niente per una questione di semplicità.

**LEZIONE 18**

**PROTOCOLLI NOTEVOLI (NON USATI IN REALTA’)**

* Alice e Bob si vogliono autenticare. A invia a B una nonce cifrata con la chiave pubblica di B, poi B invia ad A la nonce in chiaro. Così Bob è autenticato ad Alice con freshness, perché solo Bob può estrarre. È un po' una rivoluzione rispetto ai protocolli basilari che avevamo visto. Qui ci sono due messaggi e il protocollo è iniziato da Alice, questo perché chi vuole freshness deve iniziarlo. In quello classico semplicemente Bob inviava il messaggio con la sua chiave privata, ma senza freshness. Un attaccante potrebbe costruire un messaggio di questo genere ad hoc, ma non è un attacco data la proprietà di autenticazione di Bob. Per romperla dovrebbe spacciarsi per Bob. E nonostante la nonce sia in chiaro, quindi un attaccante potrebbe intercettarla e inviarla ad Alice, ma non si spaccerebbe per Bob interamente, perché la nonce di B arriva comunque ad Alice che lo autentica.

1. A -> B: {Na}kb
2. B -> A: Na

* Un’alternativa potrebbe essere quella di A che invia a B la nonce in chiaro, e B che gliela reinvia cifrata con la sua chiave privata, questo somiglia più al protocollo basilare. Essi sono equivalenti perché entrambi garantiscono autenticazione con freshness di Bob ad Alice, posteriormente al tempo di invenzione della nonce di Alice. È più simile al protocollo basilare di autenticazione.

1. A -> B: Na
2. B -> A: {Na}k-1b

* Alice e Bob vogliono ora autenticarsi mutuamente con freshness. Prendiamo il primo protocollo e lo doppiamo.

1. A -> B: {Na}kb
2. B -> A: {Na,Nb}ka
3. A -> B: Nb

Ma nei passi di autenticazione Bob non sa ancora con chi si sta autenticando, più che altro non è certo, dato che è quello lo scopo e il protocollo non ha ancora finito, in questo caso ci si affida a TCP/IP. Ma anche mettendo l’identità di A nel messaggio di A stesso, anche qui si ricade sull’affidabilità di TCP/IP però, perché si sa che mettere un’identità non è ‘sicuro’ (esplicitazione).

Ma questa autenticazione deve essere fatta ogni volta? No, vediamo come poter instaurare un segreto alla prima autenticazione, in modo da evitare di rifarla, ma ripresentando il segreto (che dovrà essere ogni tanto rinfrescato) ci si ri-autenticherà. Tramite Needam-Schroder.

**NEEDHAM-SCHRODER**

Come nel browser bisogna mostrare id di sessione al server per ogni richiesta, qui gli serve una chiave di sessione da usare ogni volta. Se scegliessero la nonce di B inviata da A a B alla fine del protocollo in chiaro, non sarebbe una buona idea, perché appunto intercettabile da DY. Basta che A invii la nonce di B con la chiave pubblica di B. Così si ottiene mutua autenticazione con freshness, e segretezza della concatenazione delle nonce. A e B hanno così un segreto condiviso. Questo è il protocollo **Needham-Schroder**, a chiave asimmetrica. Ogni tanto si dovrebbero rinfrescare le nonce, le chiavi di sessione. Non si sta parlando di certificati perché si sta supponendo che la cifratura funzioni e che le chiavi pubbliche appartengano ai proprietari.

Lowe dopo circa 18 anni disse che questo protocollo aveva almeno un attacco. Il problema stava nell’approccio informale, necessitavano dei metodi matematici e formali, cercando correttezza e completezza con dimostrazioni.

**ATTACCO DI LOWE**

Un attacco potrebbe essere se Alice vorrebbe iniziare il protocollo con l’attaccante, C, Eve. A invia quindi a C la sua identità e la sua nonce con la chiave pubblica di C. Così C può mandare {A, Na} cifrato con la chiave pubblica di B. B, quindi, pensa di parlare con A, ancora non è un attacco. B a sua volta crea {Na,Nb} cifrata con la chiave pubblica del destinatario, cioè A. C intercetta ma non può farci nulla, e la manda ad Alice che trova la sua Nonce e pensa di parlare con C, ed è vero (non è un attacco), autenticandolo, e prendendo per fede la nonce di B come la nonce di C perché non può riconoscerla. Poi continuando A invia a C Nb cifrata con kc. E C invia Na cifrata con kb a B. Prima non c’era attacco, ma con quest’ultimo passo B pensa che il suo interlocutore sia A, quando è C, ed è un attacco alla proprietà di autenticazione. È l’apprendimento di Nb di C che porta all’attacco, al fatto che Bob autentica Alice mentre invece è C.

1. A -> C : {A,Na}kc
2. C -> B : {A,Na}kb
3. B -> A : {Na,Nb}ka
4. C intercetta e rispedisce ad A
5. A -> C : {Nb}kc
6. C -> B : {Nb}kb

Questo protocollo non è pensato per il modello di attaccante Dolev-Yao (unico attaccante che prende tutto il potere degli attaccanti in rete). In realtà è ordinario usare prodotti per cose diverse rispetto a quelle per cui sono stati progettati.

L’exploit, lo sfruttamento della vulnerabilità sta quando Eve costruisce un messaggio con Na e Nb e messaggio importante, inviandolo a Bob, il quale pensa di star parlando con Alice.

Cambiando le assunzioni, pensando B come General Attacker (attaccante per i fatti suoi), senza colludere con DY, ecco che salta agli occhi che c’è la violazione, l’attacco quando B riceve la nonce di A, che invece A pensava di usare per parlare con C. Quindi importante definire un attacco in base al modello di attaccante. Bob così può spacciarsi ad Alice come C.

Come si potrebbe tramite un fix aggiustare l’attacco di Lowe (e di conseguenza quello col GA)? Ci possono essere vari fix.

**FIX ALL’ ATTACCO DI LOWE**

Immagine che contiene testo, Carattere, schermata, bianco

Descrizione generata automaticamente

Il primo non è un fix perché B potrebbe non togliere la cifratura interna con la chiave privata di A, e cambiare solo la cifratura esterna cambiando la chiave pubblica da C a B. B riceverà {A,Na} con la chiave privata di A e poi la pubblica di B, pensando di averlo ricevuto da A.

Il secondo dipende anche dal primo, ma è un fix perché questo messaggio informa Alice della presenza di qualcun altro dato che prova a decifrare con la pubblica di C, ma non capirà nulla e farà abort; quindi, dedurrà che non era un messaggio di C. Mentre Alice si aspetta C.

Il terzo è un fix perché quel passaggio di prima è irrilevante.

L’ultimo non è un fix perché l’attaccante può crearlo da 0 mettendo dentro {Charlie, Bob}. Abbiamo già notato come il fix sta nel messaggio 2, perché è qui che Alice viene gabbata.

Il quarto è un caso particolare in cui inserire l’identità funge da autenticazione, ed è inattaccabile per via della cifratura, perché C non può cambiare il messaggio. È proprio questo il fix scelto da Lowe, perché più leggero, con uno strato in meno di cifratura. Funziona perché l’identità del mittente è protetta dalla chiave pubblica. Chiunque avrebbe potuta crearla, ma se il messaggio fosse malevolo, allora i contenuti non sarebbero affidabili, ma se il ricevente si fida di questo messaggio che è malevolo, l’attacco non c’è, l’attaccante non ha ancora conoscenza della nonce di B e non può nemmeno intercettare, decriptare e conoscerla. D’altro canto, se il messaggio è genuino e il ricevente si fida, questo risolve l’attacco di Lowe perché Alice vede la presenza di Bob. Un messaggio malevolo verrà invece rigettato da Alice.

**LEZIONE 19**

**PRINCIPIO DI ESPLICITA’**

Un altro livello di perché, si può porre chiedendosi cosa Alice non è riuscita a capire, quali informazioni non ha captato dal messaggio ricevuto e che hanno portato all’attacco. Bisogna infatti ricordare che l’etichetta del mittente non serve a nulla, esso deve essere garantito dal messaggio inviato, poiché ha un patrimonio informativo. Il messaggio 2 per esempio non dice chi ha estratto la nonce, non è abbastanza esplicito, non ha patrimonio informativo. Alice può solo fare guess, è stata fregata per questo, nel caso di Needham-Schroder era significativo specificare le identità nel messaggio. Quando l’identità di chi scambia messaggi è importante, allora va esplicitata.

**WOO-LAM**

Supponiamo ci sia un sistema simmetrico e di volere autenticare Alice con Bob. Supponendo che tutti abbiano una chiave a lungo termine, ma che non ci sia ancora una chiave di sessione. Allora ci sarà un aiutante della comunicazione, il TTP, che possiede un repository di tutte le chiavi a lungo termine di tutti gli agenti. Serve una dose di fiducia ulteriore nel server. Nei server odierni le password sono possedute in formato hash, qui invece TTP sa le chiavi in chiaro.

Immagine che contiene testo, Carattere, calligrafia, schermata

Descrizione generata automaticamente

1. Alice nel primo step si manifesta a tutti, senza ricercare privacy.
2. Bob tira fuori una sfida (nonce casuale) ed è l’unica cosa che può fare, perché non potrebbe cifrare né con la chiave a lungo termine di Alice, né con la sua perché nessun altro la conosce, e non ha chiave di sessione. La nonce è in chiaro, ma qui non importa della segretezza di essa, basta ricordare i protocolli basilari con autenticazione e freshness che mandavano la nonce in chiaro, perché si presupponeva poi una cifratura seguente del destinatario.
3. Alice può solo cifrare il challenge con la sua chiave e la invia a B.
4. B non può farci nulla, ma si può affidare alla TTP, e glielo invia perché vuole trovare match tra il primo messaggio che ha ricevuto e la nonce cifrata (esplicita l’identità di Alice), il tutto cifrato con la chiave di B. La cifratura esterna è necessaria perché un attaccante potrebbe rovinare l’integrità tramite la concatenazione affiancando un’altra identità a quel crittotesto, o quell’identità a un altro crittotesto. Se lo rovina, TTP fa abort (non trova match), ma non c’è un attacco di autenticazione. Infine, il TTP che legge B dal protocollo di trasporto (supponendolo sicuro, ma se non lo fosse TTP farebbe decryption ottenendo un nosense e inviandolo a B che farà abort), fa query della chiave di B e la trova, decifra e scompone in due componenti: va nel DB e cerca la chiave della prima componente per decifrare la seconda e tira fuori la challenge. Se ci fosse stata un’alterazione dell’etichetta mittente, cioè A, al primo step di decryption non avrebbe capito nulla. Se non c’è, le componenti sono comprensibili e la prima componente è allora affidabile.
5. La nonce decifrata viene inviata a Bob cifrata con la chiave di Bob. Nel caso in cui non fosse cifrata, l’ultimo passo sarebbe uguale al 2 e non riusciremo a distinguere gli step. Se non fosse cifrata, chiunque potrebbe prenderla dal passo 2 e inviarla in chiaro a Bob come replica, skippando i passi intermedi col TTP e Alice si autenticherebbe con Bob senza i dovuti controlli. In generale nei protocolli tutti i passi devono essere diversi per evitare di bypassare operazioni.
6. Alla fine, Bob decifra, e fa matching con la nonce inviata da lui ad Alice, e se uguale autentica Alice. B è sicuro di aver ricevuto il messaggio da TTP perché solo esso ha la chiave di B a parte B stesso.

Qui come in tutti i protocolli giocattolo, l’autenticazione si ottiene alla fine, prima si hanno solo ipotesi. Per un eventuale attacco, quindi, bisogna gabbare B sostanzialmente alla fine.

**ATTACCO A WOO-LAM**

Immagine che contiene testo, Carattere, schermata, calligrafia

Descrizione generata automaticamenteUna semplice intercettazione del ManInTheMiddle e modifica del messaggio, qui porterebbe solo all’abort di TTP o di B in base alla modifica di A o Nb, quindi non un attacco alla proprietà di autenticazione.

C fa un attacco di sessione parallela, avviata con B. Mentre B non riesce a distinguere una sessione dall’altra, perché le garanzie sono date dai messaggi.

Alice potrebbe anche essere offline, il suo nome è speso a piacere da parte dell’attaccante. Il gioco viene vinto quindi su quel particolare challenge, e ciò implica che la selezione del challenge è fondamentale, non la segretezza, poiché una modifica della nonce porterebbe a un abort da B.

B ha abbinato una nonce ad Alice, e C la intercetta. C butta via la nonce assegnata a lui, non gli serve perché lui non vuole autenticarsi. Non c’è ricognizione di sessione senza session id, quindi C invia due messaggi uguali a B, che quest’ultimo non può decriptare e quindi non può vedere anomalie, cioè che sono gli stessi. B potrebbe confrontare i testi cifrati se la cifratura fosse deterministica, ma non è scalabile che egli debba fare state ogni volta, debba fare i confronti, ed inoltre è una misura discutibile perché a causa del layer di trasporto potrebbe in genere ricevere il messaggio due volte e fare abort per errore.

B li invia a TTP. Nella prima sessione B pensa che il suo interlocutore sia A, e lo mette nel messaggio. Nella seconda sessione impacchetta il crittotesto insieme a C. TTP nel primo messaggio non riesce a fare match, perché non riesce a decifrare il crittotesto, provandoci con la chiave di A. Nel secondo messaggio però la nonce sarà decifrata e andrà a buon fine. TTP invierà a B una nonce strana, quella del primo messaggio, e B tramite questo non autenticherà nessuno, e un’altra nonce che B ha associato ad Alice, ma che stava sulla challenge di C, quindi B ha autenticato C come A. Questo perché B non riesce a distinguere la sessione, e crede che quella di C non sia andata a buon fine (facendo abort di questa sessione), dato che decryptando la nonce ricevuta da TTP ottiene quella associata da A.

Dovrei riuscire a fare un fix sul progetto senza inserirne un altro, come il session id, poiché anche qui il problema sta prima.

È importante notare come B abbia solo dedotto dal messaggio con la nonce buona che fosse di A. Per fixare questo attacco basterebbe che TTP aggiungesse nel messaggio 5 alla nonce, l’identità la cui chiave TTP ha usato per decifrare il challenge cifrato. Quindi la soluzione sta solo al passo 5. In questo caso B noterebbe la nonce assegnata ad A con l’identità C, facendo abort, e non notando autenticazione di A.

**AUMENTO DI RISCHI DI ATTACCO**

SET è un protocollo ormai morto, il cui obiettivo era quello di mettere in sicurezza il commercio elettronico. Ha dimostrato che SSL non era adatto a questo scopo. SSL (si è evoluto fino a TLS 1.3) è infatti un protocollo per distribuire una chiave di sessione, ma il fatto che due condividono una chiave di sessione non implica che possano fare scambi elettronici. I protocolli di oggi che hanno sostituito SET, si appoggiano su HTTPS, quindi su SSL. Si può notare come la complessità sia aumentata esponenzialmente. Se un protocollo di 6 pagine (NS) nasconde tante discussioni e difetti, quanti se ne potrebbero fare in uno da 1000 (SET)?

**ALTRE FORME DI SICUREZZA**

Si vuole mettere un altro layer di sicurezza a un altro livello, tra trasporto e applicativo (TCP/IP). Questo perché le reti di calcolatori vennero inizialmente ideati solo con proprietà funzionali, senza sicurezza. Per fare ciò abbiamo visto steganografia e cifratura. Un’alternativa è rivedere i protocolli di rete con un’occhio alla sicurezza, infatti IPSEC sta per IP Secure. È una suite di protocolli (AH, ESP) che distribuiscono una chiave di sessione e questa la usa per cifrare i pacchetti IP. Un’alternativa alla cifratura è la MAC (Message Autentication Code), primitiva crittografica che ottiene autenticazione e integrità, similmente a una firma digitale, ma con l’utilizzo di una chiave simmetrica, e quindi funziona fra due interlocutori che la condividano. Di fatto utilizza l’hash che contiene la chiave simmetrica.

Chaffing & Winnowing è una forma di steganografia, la quale prevede di ottenere segretezza in questo modo: condividono chiave, mittente spedisce messaggio con MAC, e anche tante coppie fasulle (messaggi+MAC), il ricevente riuscirà a trovare la coppia giusta perché è l’unica su cui la MAC va a buon fine. L’obiettivo è la segretezza senza misura di encryption (c’è solo MAC e hash). La segretezza ottenuta con l’encryption è invece una forma di non deducibilità. Qui invece il messaggio è in chiaro, ma è un ago in un pagliaio ed è indistinguibile dagli altri, ottenendo segretezza in forma di indistinguibilità. Avrebbe problemi di scalabilità ed eventualmente sfruttato per DDoS. Quindi la cifratura non è l’unico mezzo di sicurezza.

**LEZIONE 20**

**IPSEC**

IPSEC sta sotto ai protocolli di livello applicativo, cioè se si ha HTTP con sotto IPSEC si è apposto. Guardare pro e contro dalle slide. SSL o TLS stanno sotto HTTP e lo rendono HTTPS. In IPSEC si lavora adesso con pacchetti, ispessita con crittografia. AH, sta per Autentication Header, ESP sta per Encapsulated Security Payload che fa la cifratura, IKE (aiki) sta per Internet Key Exchange che è una variante di DH per scambiare le chiavi di sessione supponendo un canale non sicuro, le quali servono poi a livello più basso per AH e ESP. Anche se darebbe sicurezza a livelli applicativi soprastanti, è meglio aggiungere sicurezza a più livelli per aumentarne la possibilità di efficacia.

Dietro ogni protocollo ci si aspetta delle regole, una policy. IPSEC ha diverse modalità di funzionamento e ciò presuppone una policy istituzionale nella quale voglio applicare IPSEC, che determina la variante di IPSEC. Questa policy prende il nome di Security Association (SA), ed è una associazione unidirezionale fra mittente e destinatario. Una SA ha almeno 3 campi: un ID di policy che prende il nome di SPI (Security Parameter Index); l’indirizzo IP di destinazione al quale la SA verrà applicata; identificatore del protocollo (AH o ESP).

Se ci sono più policy, più SA, come si fa a decidere quale gestire? Adesso si parla di nodi della rete, poiché si è scesi a livello architetturale. Ogni nodo fa la sua parte nel routing, e se un nodo vuole essere IPSEC compliant, deve conoscere la policy (SA) e deve poterla implementare, perché un nodo tradizionale non è detto sia IPSEC compliant, deve imparare un nuovo protocollo. Come fa a conoscere e individuare la policy? Esso mantiene un SAD (Security Association Database), che è un DB che gli permette di conoscere e gestire varie SA.

Ogni entry del SAD, oltre ai campi del SA, mantiene 3 campi aggiuntivi puramente funzionali. Quindi ogni nodo mantiene in totale 6 campi per essere IPSEC compliant. Come funziona quindi il protocollo? Il nodo ha il suo DB e sceglie il SA in base al pacchetto che riceve, le istruzioni di scelta stanno quindi nel traffico. In altre parole, il traffico IPSEC ha un ID, l’SPI, che viene letto dal nodo il quale fa una query nel DB.

Ma un attaccante cosa potrebbe fare? Cambiare l’SPI del pacchetto, facendo un downgrade della policy. Questo è un attacco all’integrità e si potrebbe risolvere con la MAC (simmetrica e con hash).

**AH (IPSEC)**

Il protocollo AH si basa su un header, frammento aggiunto a ciascun pacchetto IP. Esso è finalizzato ad autenticazione e integrità (andare a rivedere perché sono insieme). Si necessita che mittente e destinatario abbiano una chiave e se la scambino tramite IKE. Ma se i nodi sono distanti è poco scalabile, come si risolve? Essendo AH un protocollo di sicurezza, mittente e destinatario prestabiliscono una chiave condivisa, e si scambiano pacchetti e mettono qualsiasi cosa vogliono. L’obiettivo è quello di permettere ai nodi di non vedere il payload per segretezza, ma di vedere a chi lo devono spedire, poiché essi devono fare routing e glielo si deve permettere. Questo grazie a misure di crittografia del mittente che permetteranno di mantenere le 3 proprietà di sicurezza, permettendogli comunque l’inoltro fino alla ricezione del destinatario. L’obiettivo di AH è quello di prevenire replay attack e IP spoofing (cambio di mittente, che sta nel payload cifrato). I nodi di per sé sono strutturalmente man in the middle, si spera non siano malevoli. I replay attack si prevengono con freshness, tramite un altro meccanismo.

In AH è importante considerare l’SPI, il numero di sequenza del pacchetto e la parte di autenticazione, che è la MAC del pacchetto contenente questo AH, ovvero del payload. Qualunque alterazione è notata dalla MAC, in questo caso il ricevente lo butta via in seguito alla verifica del MAC. Quindi un pacchetto passa come oro colato o viene buttato dal ricevente.

Come si prevengono i replay attack? Il ricevente scarterà pacchetti troppo vecchi, ripetuti o alterati. Il pacchetto arrivato viene accettato solo se è dentro la finestra, è nuovo e non è alterato. È un meccanismo a finestra tradizionale, ma il nocciolo sta nella MAC, che controlla qualsiasi alterazione del pacchetto, anche una replicazione del pacchetto o spacciarne uno vecchio per uno nuovo. Quindi il fatto che il ricevente incrementa sempre il sequence number (la finestra entro la quale accettare), è la misura di freshness, il che è una funzione di integrità, così si evitano replay attack.

Ma se un nodo non fosse IPSEC compliant, come potrebbe fare? Nella modalità tunnel (VPN), a differenza della modalità trasporto, si aggancia un nuovo header IP all’header AH, e quello originale si cambia di posizione. Questo per una retrocompatibilità con IP perché, se un nodo IP vuole partecipare al routing in maniera benevola può farlo. Come si fa un tunnel? Agganciando quell’header nuovo al pacchetto, ma a chi appartiene questo IP?

Alice e Bob vogliono un tunnel tra due reti differenti per comunicare in modo sicuro (si stanno fidando di eventuali insider, dato che non c’è IPSEC). Il router però è IPSEC compliant, e aggiunge il nuovo header IP del router destinatario. Il pacchetto è stato inoltrato nel mezzo da tanti nodi, ma tramite la MAC è possibile essere sicuri che sia sicuro. Fra A o B e il loro gateway il pacchetto è nudo, il tunnel non c’è. Tramite questo meccanismo i router in mezzo possono anche non essere IPSEC compliant.

**ESP (IPSEC)**

Qui il concetto di trasporto o tunnel non cambia. Nasce per la segretezza, ma attenzione a non cifrare l’SPI. Il pacchetto ESP è costituito dall’SPI, numero di sequenza (compongono header ESP), payload tradizionale, padding e sua lunghezza, next header e parte di dati di autenticazione opzionale (è una ridondanza). L’encryption non sarà applicata all’header ESP. La modalità trasporto non è vincolante del destinatario, il routing si può fare, però il payload può essere decifrato solo da chi ha la chiave. Il tunnel si fa come in AH: l’header IP viene cifrato (non in AH) e il tutto incapsulato con un nuovo header IP, che sarà l’IP del bounder di router remoto, l’unico a possedere la chiave per fare decryption dopo aver strippato l’header ID nuovo.

**COMBINAZIONE TUNNEL**

Ci sono varie combinazioni di SA. Ad oggi in realtà ha senso combinare più di due modalità di trasporto. Può servire combinare più tunnel, ma ogni tunnel deve avere un inizio e una fine. Il tipo 1 è punto-punto e può essere AH trasporto, ESP trasporto o entrambi (con due tunnel). Il tipo 2 è VPN, la sicurezza è tra intermediari, in AH o ESP. Il tipo 3 combina tipo 1 e tipo 2 con sicurezza punto-punto, e il tunnel tra i due security gateway; per la zero-trust (o principio di difesa in profondità), se fallisce una misura ce n’è un’altra. Il tipo 4 è come il tipo 1 con un tunnel tra Alice e il suo bounder (binder? Bound? Bouncer?), usando IPSEC due volte.

IKE è a sua volta una suite di due protocolli: Oakley, una variante di DH; ISAKMP che crea chiavi di sessione usando la chiave iniziale per poterle rinfrescare, prologo che si vedrà su SSL (Pseudo Random Function PRF). Si genera una nuova chiave sfruttando il seme della vecchia chiave e l’entropia dalle nonce fresche.

**LEZIONE 21**

**VARI ATTACCHI E DIFESE REALISTICI**

Il rilevamento delle intrusioni non vuol dire avere firewall perché un processo alieno potrebbe introdursi anche tramite una pen-drive o da tutti gli altri livelli architetturali. È una disciplina molto grande e consolidata, antica. Esiste un Working Group a livello internazionale. È un processo vero e proprio. Il firewall non si occupa di intrusioni intese in questo modo. Per quanto sia avanzata, non potrà mai scremare perfettamente falsi positivi o falsi negativi. È un ambito non esatto, ma euristico, in cui le risposte assolute non esistono. Worm e malware non sono per forza intrusioni, ma possono essere installati tramite esse. Sono differenti dai Denial of System (DoS), i cui risultati sono immediati. Ci sono diversi tipi di DoS, sono attacchi alla disponibilità del sistema e possono essere alle risorse, alla memoria e alla banda. Un esempio di attacco banale sarebbe quello in cui un router fa da broadcast in una LAN. Per DDoS si intende Distributed Denial of System. Inoltre, spesso le richieste possono essere fatte senza autenticazione.

**DOS**

L’attacco più comune oggi giorno di negazione del sistema è quello distribuito, DDoS, in cui si sfruttano tante macchine ‘zombie’ che attaccano una vittima. Un esempio ne sono le telecamere che sono costituite spesso da firmware con vulnerabilità che permettono all’attaccante di impossessarsene e di fare richieste http, potenzialmente creando un esercito di telecamere che fanno richieste. Gli attacchi DDoS sono quelli che fanno più danni reputazionali.

Come ci si difende? Si chiudono le connessioni, ma ovviamente è sconveniente; un approccio potrebbe essere quello di filtrare il numero di connessioni; un’altra euristica famosa è cookie transformation, tramite la quale il ricevente non si concede subito: il client manda la richiesta, il server manda il cookie che costa poco o magari ce l’ha già pronto in batch, e il client dovrà esibire la storia della sessione ed eventualmente un hash o qualcosa per trasformare il cookie ricevuto, che anche se semplice per il client, ciò rompe il banale loop di un eventuale DoS, c’è comunque un lasso di tempo che si perde. Ciò inoltre necessita anche un client attivo.

Che differenza c’è tra intrusione e negazione del servizio? Il DoS ha effetti temporanei, mentre l’intrusione ha effetti persistenti, il DoS si nota subito, mentre le intrusioni potenzialmente mai, il DoS blocca le risorse, mentre le intrusioni usano in maniera illecita le risorse.

**INTRUSION DETECTION SYSTEM (IDS)**

In generale un’intrusione è un accesso o un innalzamento dei privilegi rispetto a quelli leciti. Le tecniche sono violazioni di password, intercettazioni di informazioni, uso di combinazioni di SW nocivo, pen-drive, social engineering.

L’IDS è una disciplina che non può fare a meno dei falsi perché non si riesce ad avere una caratterizzazione univoca di un’intrusione. Per questo si fa molto uso di AI perché gli algoritmi deterministici non vanno bene. Tuttavia, gli strumenti più tradizionali di IDS non usano machine learning.

I meccanismi basilari sono i log. Un SIEM è uno strumento che usa tecniche IDS in maniera più spinta e organizzata, spesso con IA, loggando tutte le trasmissioni per trovare correlazioni che possano significare un attacco. Fare logging è facile, il problema è fare le decisioni. Inoltre, fare log costa molto perché si tratta di volume di dati. Dati i log come si può fare detection, e differenziare quindi un comportamento malevolo da uno benigno? Con osservazioni statistiche come il comportamento medio nel tempo, anche se by design portano dei falsi.

Gli strumenti di log sono i record nativi che tutti i SO odierni hanno; record specifici per il rilevamento che danno informazioni ulteriori, come le azioni elementari per ogni operazione. Un esempio storico di record specifici sono i record di Denning costituiti da soggetto, azione e oggetto, chi fa cosa su che cosa, se ci sono delle eccezioni; quali risorse vengono utilizzate; e timestamps. È una entry nel log. Ad oggi i record usati sono molto più complessi.

Dato un log come si fa rilevamento? Si possono usare tecniche assolute indipendenti dal passato, come il controllo a soglia già usato per esempio per contrastare l’attacco dizionario o il brute-forcing della password. Il numero della soglia nasce dalle ricerche sull’intrusione, che normalmente va per brute-forcing. Gli obiettivi tra anti-bruteforce della password e intrusion detection sono diversi anche se sono di comune ispirazione: nel primo caso si blocca completamente l’accesso, nell’altro si logga per poi fare detection in base alla soglia scelta. Le tecniche dipendenti dal passato, si hanno campionando il comportamento passato e assumendo che il futuro sarà come il passato. Infine, ci sono quelli statistici che creano un modello statistico del comportamento di un utente, e a regole come Snort. Tra quelli statistici ci sono le deviazioni standard (comportamento medio) e quello operativo (a soglia).

Per ogni parametro da controllare, ci sono modelli di rilevamento statistico più opportuni di altri. Anche il tempo dall’ultimo login è un parametro, controllato meglio con il controllo a soglie (operational). La frequenza di esecuzione di un comando con la deviazione standard e guardare cosa rileva. Studiare tutta la tabella. È importante capire che una volta fatta una detection, poi bisogna ispezionare perché si potrebbe trattare di un falso.

Nel rilevamento a regole c’è un grosso database di regole. Ci sono importanti istituzioni che espandono le linee guida di gestione di un’intrusione.

Per la prevenzione si dovrebbe limitare l’automazione, usare honeypot. IMS (Intrusion Management System): in generale l’intrusione va contenuta, rimossa, riassorbita (recovery, ripristino delle condizioni) e punita.

**MALWARE**

Un bug è un aspetto funzionale inatteso, la feature è progettata, voluta. Di contro un SW nocivo ha il compito di fare danno. I bug possono essere sfruttati come punto di accesso o di ispirazione per i malware; oggi si chiamano vulnerabilità (quando il bug viene sfruttato da un malware) per questo. Quando si parla di malware gli elementi fondamentali sono il payload (carico) e meccanismo di propagazione. Tra i SW nocivi ci sono quelli indipendenti, come gli zombie (vedere attacchi DDoS con telecamere) e i worm. Quelli dipendenti da programmi ospitanti sono i virus e i trojan, bomb logic e trapdoor. La trapdoor o backdoor è un punto di accesso sul retro lasciato lì volontariamente (certamente non è bug), il cui obiettivo funzionale doveva essere beta testing (per saltare l’autenticazione per esempio) o un salto a una determinata parte del codice. La logic bomb è un frammento di codice che aspetta l’istante dell’attivazione che dovrebbe corrispondere a un determinato trigger, in genere ha fini distruttivi del SO. Il trojan è apparentemente inutile, ma in fase di esecuzione viola la sicurezza. Zombie è un programma nocivo che sfrutta una macchina remota già violata per lanciare attacchi che difficilmente portano alla riconduzione dell’autore dello zombie.

La differenza tra worm e virus è che il worm è un programma indipendente, si trasmette ed è più pericoloso, in genere non richiede intervento umano (ha carico e propagazione). Il più famoso è il worm di Morris. La definizione di bug nel worm di Morris potrebbe essere stata una trovata per la sua difesa. Leggere dalle slide. L’attacco a cifrario noto (deterministico, es. di Cesare) consiste nel fare brute-force fino ad ottenere una password che cifrata è uguale al testo cifrato, simile all’inversione di un hash (crypt3). BOF sta per BufferOverFlow. Se non fosse riuscito nell’attacco al cifrario, avrebbe usato BOF. In alternativa la trapdoor.

Il virus è una banale manipolazione di un file, vi si attacca. Basti pensare che modificare un solo bit di un file eseguibile, lo fa crashare. Un virus polimorfe cambia la sua firma, che è una sorta di suo ID. Esso quindi si identifica tramite la sua firma. Gli serve per capire dove è stato per fare più danno possibile, ma è il suo lato negativo per la detection. L’antivirus cerca i virus in base alle firme del virus.

**LEZIONE 22**

**VIRUS**

Un virus si attacca a un programma ospitante, il cui obiettivo è l’alterazione. Originariamente i virus non erano usati per esfiltrare dati, ma perché sui dati digitali si teneva ben poco di importante. Ad oggi invece è totalmente diverso, i dati digitali stanno ovunque anche per IoT, casa, macchina, smartphone. Microsoft ha ad oggi più di un antivirus, è un security center, ha antivirus, IDS, firewall e molto altro. Scrivere virus, rompere un qualsiasi eseguibile è molto semplice. Mentre l’exploit di una vulnerabilità è più complesso e moderno.

Cosa è attaccabile su Office? Le macro sono pezzi di codice che Office esegue, che possono essere installati anche dietro una cella. Windows ha messo delle soglie per l’eseguibilità delle macro, adesso sono disabilitate di default, per questioni di sicurezza dato che dietro una macro può esserci codice malevolo. I macrovirus sono virus che si celano in macro di altre app. Tra i SW nocivi ci può essere relazione causa-effetto, come quella di installare un trojan tramite backdoor.

Il virus nel codice prende un file a random, e se questo non ha la firma, piazza un danno tramite una bomba logica, ma in genere il danno viene fatto senza un trigger, subito. La posizione del virus è frutto del continuo scontro tra attacco e difesa, perché lo scopo del virus è quello di non farsi scoprire. Inoltre, un virus aumenta la taglia del file, quindi di contro c’è stata una difesa con confronto tra taglia standard e reale. A sua volta i virus si sono evoluti, comprimendo il file target in funzione della dimensione del virus stesso e quindi occupando meno spazio nel file. Prepend nel codice indica l’aggiunta del codice del virus nel file. Il fatto che esso si preappenda implica che il virus si attacca e può quindi riprodursi, non viene eseguito una sola volta, la prima in cui fa danno al file, ma inoltre tramite la prepend può fare ulteriore danno in altre istanze del virus stesso.

Un esempio di virus è brain, che secondo la storia ufficiale (a giudizio), rinomina il volume del disco in BRAIN, e alcune versioni cancellano frammenti dell’HD. Il virus si mette in una porzione della memoria e la setta in modo che esso non verrà toccato, e poi setta la posizione 19 del vettore degli interrupt per gestire letture da disco come proprio indirizzo per farsi eseguire. Quindi gestirà tutte le letture. Mette una firma banale, 1234, ma la sua furbata è dove va a mettersi, tra 5° e 6° byte, ciò era difficile da scoprire. Marcherà i settori infetti come danneggiati (non usati poi dal SO) fino a infettarli tutti.

**RIMOZIONE SW NOCIVI**

Di base sarebbe buono ma troppo dispendioso scansionare tutta la memoria di massa. Antivirus; sistemi immuni che in realtà è una distribuzione sulla rete degli antivirus, in formato distribuito diciamo, ogni sistema ha una sonda che invia informazioni a una macchina centrale che controlla; SW sentinella che localmente bloccano esecuzione di codice nocivo. Un antivirus individua, poi lo identifica (capisce che virus è) e lo elimina, ma quest’ultimo passo non è banale, poiché l’antivirus deve conoscere la circoscrizione del virus perché rischierebbe di eliminare parti non malevole di file. Si distinguono 4 generazioni di antivirus. Nella prima si fa una scansione dei file alla ricerca di firme prese da un DB, o ricercare variazioni significative di file. I virus della seconda generazione usano euristiche, cioè tecniche non sicuramente funzionanti, come ricerca di frammenti di codice tipicamente associati a virus (con eventuali falsi negativi e positivi) e ricerca di integrità. La terza ha implementato il fatto di avere l’antivirus sempre in memoria, eseguito, e aggiunge altre euristiche. La quarta è un re-make che ha anche la capacità di regolare il controllo d’accesso per limitare la propagazione del virus. Una tecnica avanza di antivirus è quella di usare macchine virtuali senza rete (air-gapping).

**FIREWALL**

Sono una misura di sicurezza di natura preventiva, se il malware è già preso il firewall non può fare nulla. Esistono anche i web-application firewall. I firewall possono lavorare a più livelli dello stack, più sta in alto più è costoso. Se l’intrusione è dentro non si può più pensare al firewall, ha fallito, ha senso in un sistema hardenizzato. È un dispositivo che fa enforcing di una policy per controllo del traffico tra due reti, due ambienti diversi, con regole e struttura diverse. Per esempio, tra LAN e rete esterna. www è un server, quindi, c’è da gestire la 80 o la 443. I firewall a livello di rete sostanzialmente aprono e chiudono le porte. Ci possono essere più di un firewall. Che requisiti deve avere un firewall? Slide (bastion host = macchina hardenizzata). Se integro il firewall alla macchina del www, non sarà una scelta ottima, perché si porta i suoi potenziali exploit, per il principio della separazione degli obiettivi.

Slide sulle funzionalità. Di per sé non rileva le intrusioni, ma gestisce le porte, però possono essere integrati. Al firewall interessa più l’indirizzo di destinazione che quello di origine. Un IDS invece è un portiere più forbito, che guarda il payload. I firewall commerciali costosi in realtà sono più che un firewall (in più funzionalità 4,5,6), si fanno atti di fede perché c’è molto altro. Si possono applicare due regole (infatti le policy trovano molta applicazione nei firewall, con l’ordine di priorità per evitare contraddizioni), default denay dove tutto ciò che non è permesso allora è vietato. I limiti del firewall sono che non protegge dagli insider, e non guarda il payload, potrebbe degradare le prestazioni e non proteggono da attacchi ancora non documentati. Un firewall di base non fa altro che filtrare pacchetti, potrebbe anche fare stato. Firewall personale da slide, security defender di windows lo fa. Slide packet-filtering. Oggi c’è più sensibilità nel filtrare pacchetti in uscita, perché si potrebbe avere per esempio degli zombie dentro il perimetro delineato dal firewall, è un aspetto che sfruttavano le reverse-shell.

Le regole per packet-filter sono il controllo degli indirizzi di source e destination, indirizzi di origine e destinazione a livello trasporto, il protocollo IP, l’interfaccia e decide. Esemplificare due linee di firewall. L’interfaccia cambia da firewall in firewall. Slide su esempi di firewall. Non è detto che a una determinata porta ci sia sempre lo stesso servizio, ma nmap ci darà il daemon che c’è dietro, poiché il sys admin può cambiare le porte dei servizi come vuole. Quindi attenzione a servizio malevolo che bypassa firewall.

I limiti dell’IDS sono che ha falsi positivi e falsi negativi, nonostante siano poche, il problema rimane la gestione una volta identificata l’intrusione, in quelli commerciali potrebbe istruire il firewall a chiudere una connessione, una porta. Realisticamente l’IDS non va ad istruire il firewall, ma si occupa solo di detection, e non chiude connessioni perché ci potrebbero essere tanti falsi positivi dati i tanti alert. I firewall riescono a fare encoding con linguaggio di ip e porte di servizi, come FTP. I firewall di contro sono poco espressivi, spoofing IP se non c’è hardenizzazione di IP.

Stateful inspection firewall concede l’ingresso dati solo sulla specifica porta richiesta osservando lo stato della connessione, cioè, è una regola imposta che vi sia quello stato. Slide di DMZ. Più i servizi sono interni a questa architettura a layer, più sono interni. Se i servizi sono interni e dovranno essere esposti all’esterno, il firewall che sta davanti deve gestirli esplicitamente. Esempio di VPN, VPN + DMZ, Applicazione 4 è IPSEC in modalità tunnel integrato col firewall. Applicazione 5 è IDS di rete distribuito, su ciascuna DMZ.

**LEZIONE 23**

**SSL**

Ricordare che qui gli interlocutori sono il web-server e il browser del client. HTTPS è il protocollo ormai più utilizzato oggigiorno, è un potenziamento di HTTP quando sopra SSL. TLS è SSL aggiornato. I protocolli sicuri sono usati su e-mail, pagamenti e transazioni. Guardare documentazione di HTTPS su RFC 2818, ci dà tutte le informazioni sul protocollo delle quali pochi si riferiscono alla sicurezza. HTTPS è esposto sulla porta 443; cifra cookie, contenuti, form e HTTP header. L’header speciale HSTS è stato fatto in difesa di SSL stripping. Esempi di attacchi SSL in slide, il secondo è URL spotting. Non tutti i siti ascoltano sulla porta 80 e danno la versione in HTTP però, in quel caso si potrebbe proprio cambiare il sito come nella versione 2.

In termini architetturali sta sotto i protocolli applicativi (HTTP), e sopra TCP. Fa da livello di sessione/presentazione, ma è anche inglobato a HTTPS quindi anche applicativo. Vedere in slide quanti protocolli sono in versione sicura tramite SSL.

SSL nasce nel 94. TLS sta per Transport Layer Security. La standardizzazione di SSL è avvenuta 15 anni dopo perché ci vuole molto tempo per fare ciò. TLS è la versione standardizzata di SSL.

SSL è una suite di protocolli. SSL record protocol sta sotto perché è quello che implementa la sicurezza (almeno autenticazione e segretezza). Tutto quello che sta sopra sfrutta la sicurezza da esso. È il corrispondente in IPSEC di AH e ESP. Per dare sicurezza si necessita di cifratura e quindi servono le chiavi, che sono gestite dal protocollo SSL Handshake Protocol (omologo di IKE in IPSEC). SSL Change Cipher Spec Protocol è il protocollo di sincronizzazione, il direttore d’orchestra, il clock di sistema.

**RP/ CCSP/ AP (SSL)**

SSL RP sta per SSL Record Protocol. Deve applicare la cifratura. È composto da un Application Data che è un blocco frammentato in sottoblocchi che poi vengono compressi per ottimizzare, a ciascuno di essi ci si aggiunge MAC (per autenticazione e integrità), poi avviene la cifratura a chiave simmetrica condivisa, infine per ragioni funzionali si etichetta da fuori per gestirlo dato che è cifrato, facendo un append dell’header SSL Record. Il MAC è un hash annidato nel caso di SSL (normalmente è un semplice hash). L’hash più interno ha il padding (per ottenere il numero di bit necessari per quella funzione hash), un numero di sequenza per riconoscere, il tipo di compressione avvenuta precedentemente, lunghezza del frammento e infine il frammento vero e proprio, all’inizio la MAC write\_secret è una chiave di sessione, viene concatenata con un padding e tutto l’hash seguente, e fatto l’hash di tutto. Il record header trasporta informazioni da uno dei protocolli precedenti, e ha bisogno che il protocollo di handshake abbia funzionato perché necessita la chiave di sessione. Contiene il tipo di contenuto e altro. La cifratura usa una chiave diversa da quella di sessione usata nel MAC (MAC\_write\_secret). Per cipher suite si intende come la SA in IPSEC, è una policy.

CCSP è il direttore d’orchestra. È un momento di sincronizzazione che mette in atto le scelte delle chiavi del protocollo di handshake precedente, cioè la Cipher Suite diventa attuale, cioè le preferenze crittografiche. Decide che appena si finisce tutto ciò che si è fatto si ricomincia in quel modo, è tipo un clock di sistema. Anch’esso sfrutta sicurezza di SSLRP.

SSL AP sta per Alert Protocol. Anche questo è protetto dal record protocol. Ha 2 byte, il livello (fatal o warning) e il codice (specifica il preciso alert). Esempio di fatal: unexpected message, bad record mac (quando non funziona il MAC, informiamo l’altro che dobbiamo chiudere), problemi di decompression; esempi di warning: certificati non supportati, revocati o scaduti. Un fatal error setta Is Resumable a 0, cioè sessione non ripristinabile.

**SSL HP**

SSL HP sta per Handshake Protocol. Stabilisce un Master Secret, che è una chiave di sessione, dal quale deriveranno le altre chiavi generate tramite funzioni pseudo-random. Esso è costituito dal tipo, lunghezza e il contenuto. Il tipo è il nome che diamo al messaggio per individuare lo scopo di esso. Vedere i messaggi dalle slide (Hello request è un modo per dire “inizia tu la conversazione, il server al client). C’è un doppio scambio di chiavi perché dietro c’è Diffie Hellman (SSL HP è la sua versione concreta e funzionale). I parametri sono il corpo del messaggio. SSL usa la certificazione perché DH aveva bisogno del potenziamento con la certificazione (autenticazione) per funzionare poiché aveva l’attacco.

Nella fase 1, costituita da Client\_hello e server\_hello si stabiliscono tutte le regole. Client-Hello è un accordo a due vie (c’è una richiesta e una risposta che è l’ultimo messaggio -> ci sei? Si, ci vediamo alle 5? Ci vediamo alle 7, conta l’ultima parola), in cui ci si scambia la versione (più alta supportata, ci si prova); una nonce per la freshness; una session ID che, se è 0 vuole creare una nuova sessione (vuole un nuovo session ID), sennò chiede il ripristino della sessione (efficienza); la cipher Suite che è la lista di quale cipher e algoritmo di scambio di chiavi (DH o RSA); lista dei metodi di compressione supportati. Tutti questi campi sono preferenze del client.

Il server Hello manda la versione massima che riesce a supportare e in caso necessario più bassa del client (client invia 3 e server dice 1-2, quindi client dovrà accettare di downgradare la versione magari alla 2); random che è la nonce per rinfrescare segreti e prevenire replay attack; session id in base alla richiesta del client; Cipher Suite e Compression Method scelgono tra quelli del client. Anche quando si ripristina una sessione bisogna per forza passare dalla fase iniziale e finale di SSL (1 e 4), tagliando quelle che ci stanno in mezzo, e quindi dovranno reinventare una nonce per freshness. Cipher Suite include le specifiche dello scambio delle chiavi e la cipher spec, la Cipher Spec include almeno i primi 3 punti della slide. Key exchange Algorithm scelto tra RSA e varianti di DH. Ricordare 2 esempi di Cipher Suites.

Nella fase 2 (Autenticazione del server e scambio delle chiavi) il server manda il certificato, la sua chiave e richiesta di certificato. In SSL2 erano opzionali. Server\_hello\_done è un messaggio di conferma con firme con campi random per prevenire attacchi di replay, ottenute con RSA concatenando MD5 e SHA1.

Nella fase 3 il client manda il certificato, il secondo dei messaggi di DH e completa il messaggio server key exchange, alla fine di questi passi client e server hanno una chiave condivisa (48 byte), è la Pre-Master Secret. Certificate\_verify (che è facoltativo) firma tutto il traffico visto finora in modo che il server possa verificarlo, comporta garanzia d’integrità. Non abbiamo RP a supporto ma vogliamo costruire segreti da dare a RP. Il Master Secret sarà derivato dal Pre-Master-Secret (primo esempio di derivazione pseudorandomica di materiale crittografico da materiale crittografico noto).

Nella fase 4 si è finito e si negozia la cipher spec da utilizzare. Il messaggio finished è l’intero digest (hashing) del traffico utilizzando la nuova cipher spec. Lo scambio è a due vie. Fra la 3 e la 4 ci sono dei calcoli offline che essi fanno.

Fra la 3 e la 4 ci sono una serie di calcoli offline che entrambe le parti fanno. Un calcolo è quello di derivare il MS dal PMS. Si fa MD5(SHA (PMS || la nonce client || nonce server)) || ulteriore istanza || ulteriore istanza fino a quando non mi basta. I 48 bytes si ottengono dai 16 singoli delle 3 concatenazione. La differenza tra la 3 applicazioni sono il parametro ‘A’, ‘BB’, ‘CCC’, che vanno a dare risultati estremamente diversi in seguito all’hash. Il ruolo di salt è più che altro fatto dai random values (nonce) che allungano l’input della funzione hash.

Col Master Secret voglio n chiavi di sessione, si può usare di nuovo questo metodo? Si. Si vuole ottenere il MAC secret e la chiave per l’encryption che si chiama Write Key. Faccio la stessa cosa precedente tranne mastersecret e premastersecret, e inoltre ci sono i puntini (si possono avere più applicazioni del metodo invece che 3). Qui produco non il master-secret, ma un key block generico costituito dalle 2 cose dette prima.

Una sessione è una partita tra client e server sotto questo protocollo che definisce la cipher suite e può essere usata in più connessioni (ripristinata) per aumentare l’efficienza (non è buono fare ogni volta tutte queste scelte crittografiche). Quindi la cipher suite rappresenta e identifica una sessione. Per salvare lo stato della sessione cosa si dovrà salvare? Il Session ID, Certificato, metodo di compressione, cipher spec, master secret, se è ripristinabile. Registro solo il master secret, congelato, e poi da questo nasceranno in altri ripristini della sessione blocchi di chiavi differenti date le nuove nonce (nuove chiavi di sessioni fresche). Lo stato viene salvato alla fine della fase 4. Nella fase 1 si potrebbe ottenere il ripristino della sessione. Fase 2 e 3 vengono skippate in questo caso (perché servono per distribuire la chiave).

**TLS**

La prima versione è del 1999. È la standardizzazione di SSL 3.0. Il MAC si fa con l’hMAC (è oggi uno standard usato molto). Si usa una Pseudo Random Function per generare nuovo key block a differenza di quegli annidamenti di hash, ma si ispira a questi generalizzando. Ci sono ulteriori cipher suite e alert.

L’hMAC è parametrico sulla funzione hash, dentro c’è la chiave che è MAC\_WRITE\_Secret XOR padding, concatenato con un ulteriore applicazione dell’hMAC con la chiave XOR padding concatenato col messaggio. La differenza sta nella presenza dello XOR (non si può rompere a patto di usare chiavi fresche).

La PRF viene definita in termini di una P\_H. Prende il segreto (PMS per MS, MS per key block), etichetta (entropia, è ‘master secret’ o ‘key block’) e seme (concatenazione dei valori random, nonce). Cambia che le funzioni non sono annidate ma c’è lo XOR.

LA P\_hash prende un pezzo del segreto e il seme. Nasconde più iterazioni dell’hMAC. Il primo pezzo dell’output nasconde due iterazioni di hMAC (risultato della prima dandogli seed e secret, concatenato col seed e usato come seed della seconda hMAC con lo stesso secret). Gli altri pezzi si ottengono ripetendo queste iterazioni quanto si vuole, incrementando di una hMAC. Quindi in base a quanto voglio lunga la PRF, posso decidere quanto deve essere lungo il primo parametro P\_h.

Date su slide. Ad oggi c’è TLS 1.3, migliore in efficienza. TLS 1.2 è il primo che usa le estensioni, citate per la prima volta in TLS 1.1. Le estensioni vengono dopo il metodo di compressione alla fine di client\_hello, allunga il messaggio. Le estensioni servono per trasportare per esempio HSTS. Quindi TLS 1.1 non risolve SSL Stripping, è anteriore alle estensioni. TLS 1.2 non permette più a client e server di specificare quale hash accettare, dato che MD5 era debole (rotto nel 2004, ma RFC del 2008). Da TLS 1.2 si può specificare quale hash. La PRF di prima è stata rimpiazzata da cipher-suite-specified PRFs, cioè come fare la PRF può essere negoziato nella cipher suite, la PRF può essere definibile come parametro ulteriore nella cipher suite.