Ciao ragazzi, questi sono dei mini-riassunti dei consigli di lettura che ha dato l'anno scorso, evidentemente non sono proprio super corretti visto che alla domanda su questi ho preso 4 su 6 punti. Comunque possono esservi utili!

## Consigli di lettura 1 - How the NSA almost killed Internet - Wired

Google, Facebook, Microsoft, e altri titani della tecnologia hanno dovuto combattere per la propria vita contro il loro stesso governo.

Nel giungo del 2013 il Guardian (giornale britannico) ha rilasciato un articolo in cui si evidenziava il fatto che la società di telecomunicazioni Verizon aveva consegnato volontariamente alla NSA un database contenente tutte le chiamate sulla sua rete. L'informazione proveniva da Snowden un consulente IT che che aveva lasciato gli Stati Uniti, con centinaia di migliaia di documenti relativi a procedure segrete della NSA.

I file di Snowden hanno indicato che alcune delle più grandi aziende sul web avevano concesso l'accesso diretto ai loro server ad NSA ed FBI, dando la possibilità di avere qualsiasi informazione di ogni persona.

Il Post, e subito dopo il Guardian, hanno pubblicato un rapporto che descriveva il programma Prism che includeva diverse immagini trapelate da un PowerPoint dell'NSA di 41 slide, tra le quali venivano elencate le aziende tech che avevano partecipato al programma (e relativi loghi) e le date in cui avevano iniziato a cooperare pienamente. Tale programma prevedeva che le aziende condividessero dati a loro conoscenza quando un tribunale segreto ordinava loro di farlo, a scopo di sicurezza nazionale, specie dopo l’11 settembre (quindi la condivisione di dati è stata spesso involontaria). E' un programma autorizzato a recuperare i dati direttamente dai data center.

Per le aziende coinvolte non c'erano solo i ricavi in ballo (senza la fiducia dei clienti i giganti tecnologici non avrebbero giro d'affari), ma anche gli ideali che hanno sostenuto il mondo tecnologico dalla nascita di Internet.

Se la rete è vista come un mezzo per la sorveglianza diffusa, la paranoia risultante potrebbe influenzare il modo in cui la gente la usa. Le nazioni di tutto il mondo erano indignate dalle pratiche di raccolta dati intelligenti utilizzate degli Stati Uniti tanto che erano arrivate a richiedere che i dati generati nei loro paesi rimanessero lì dove gli americani non avrebbero avuto facile accesso. L'implementazione di un tale sistema potrebbe balcanizzare il web (es. i dati degli italiani in Italia), distruggendo la sua essenza aperta e aumentando drasticamente il costo di fare business.

Le cose si sono aggravate quando sono trapelate nuove informazioni secondo le quali non solo NSA ha ricevuto dati direttamente dalla fonte, e quindi a valle, (programma Prism), ma in prima persona e senza la conoscenza o cooperazione delle società ha intercettato una grande quantità di dati attraverso la rete, e quindi a monte: Google e gli altri grandi dell'informatica sapevano infatti di dare informazioni all'NSA ma non sapevano che gliene davano una quantità così elevata. Sfogo dell'analista di Google è che l'NSA sniffava le informazioni scambiare tra un cloud e l'altro senza crittografia senza consenso di Google (questo è un metodo a monte) e che quindi passano in chiaro.

La motivazione con cui NSA si difende è la raccolta di informazioni per contrastare il terrorismo. Ora le aziende tecnologiche non solo dovevano combattere contro i criminali informatici, ma anche contro il loro stesso governo.

Dal momento delle rivelazioni, molte aziende hanno rafforzato la loro sicurezza, finalmente implementando la crittografia dei dati sia in transito che all'interno dei data center dell'azienda.

L'anno successivo le corti hanno decretato che il programma di sorveglianza di massa è legale. Se non sei cittadino americano sei passibile di ogni intercettazione di massa.

Se sei americano ma non è individuabile la sorgente dati allora sei passibile come tutti gli altri.

## Consigli di lettura 2 - Reflections on Trusting Trust - Ken Thompson

L’articolo di Ken Thompson descrive un gioco in cui ha cercato di scrivere un programma sorgente che, se compilato ed eseguito, producesse come output una copia esatta della sua fonte.

Nella prima parte mostra il codice di un programma che ha come output il programma stesso (il programma non è precisamente un programma che si autoriproduce ma produce un programma che si autoriproduce).

Il programma che ha scritto ha due proprietà che saranno utili ai fini del suo articolo

1) questo programma può essere facilmente scritto da un altro programma

2) questo programma può contenere una quantità arbitraria di bagaglio in eccesso che verrà riprodotto insieme con l'algoritmo principale

La seconda parte mostra una modifica al compilatore C che genera tale sorgente: l'utente si fida dell'azienda che fornisce il compilatore e proprio in questa fase viene fregato.

La terza parte fa in modo di rendere tale modifica come se fosse una modifica di fabbrica, dimostrando come sia facile effettuare una modifica che sia invisibile agli utenti allo scopo di fregarli. Se la modifica non è intenzionale si può parlare di bug, se è intenzionale di cavallo di troia.

*Esempio login*

Si vuole inserire una backdoor che permette di saltare la procedura di autenticazione se l'username è lungo tot caratteri. Questa backdoor ha vita breve perché chiunque controlli il sorgente del login se ne accorgerebbe.

Modifico allora il sorgente del compilatore in modo che tutte le volte che viene compilato l'eseguibile del login venga inserita la backdoor nell'eseguibile. Anche questa backdoor ha vista breve perché uno che guarda il sorgente del compilatore lo trova.

Allora introduco una backdoor nel sorgente del compilatore che fa si che quando quel compilatore compila un altro compilatore crei due backdoor

* tutte le volte che viene compilato un compilatore vengono inserite due backdoor
* tutte le volte che viene compilato un eseguibile aggiungi una backdoor al login

Posso prendere il sorgente del compilatore e tornare alla versione primitiva e non vedere la backdoor perché questa vive all'interno del binario (non esiste nel sorgente del compilatore, nemmeno nel sorgente del login, esiste solo nel binario del compilatore).

L'unico modo per vedere se c'è una backdoor è andare a prendere il binario del compilatore e controllare il codice Assembly del binario (senza alcuna speranza di capire che c'è una backdoor in quel binario).

L’idea è quindi: mettere la backdoor nell'eseguibile, poi nel compilatore, poi nel compilatore del compilatore e così via. Qual è la versione precedente tale che si presuma non essere maliziosa?

La morale è che non ci si può fidare di un codice che non si è creato totalmente in prima persona.

Nessuna quantità di verifica o controllo del codice sorgente protegge dall’uso di codice non attendibile.

Nel dimostrare la possibilità di questo tipo di attacco, si è preso come esempio il compilatore C. Come il livello di programmi diventa minore questi bug saranno sempre più difficili o impossibili da individuare.

L’autore critica la stampa nella gestione degli hacker: le azioni eseguite sono atti di vandalismo nel migliore dei casi ed è solo l'inadeguatezza del codice penale che salva gli hacker da accuse molto gravi. Le aziende che sono vulnerabili a questa attività stanno premendo per aggiornare il codice penale. L’accesso non autorizzato a sistemi di computer è già un serio crimine in molti stati. C’è una situazione per cui da una parte stampa, televisione e film rendono eroi questi vandali, dall’altra gli atti eseguiti da questi ultimi saranno presto punibili con anni di prigione. Spesso gli hacker soprattutto giovani sono completamente inconsapevoli della serietà delle loro azioni. L’atto di violazione del sistema di un computer deve avere lo stesso stampo sociale che ha la violazione di un’abitazione.

Domande che fa porre questo articolo

* Può esistere sicurezza senza fiducia (trust)?
* Come fai a compilare il compilatore? C'è bisogno di un compilatore?
* E' nato prima il codice o un compilatore? Il primo compilatore è stato scritto in assembly, poi in C. Così si ha una catena di compilatori dove il compilatore n compila quello n-1 ecc.

## Consigli di lettura 3 - The Internet Worm Incident - Eugene H Spafford

Nel 1988 la rete Internet (allora di 60.000 nodi) è stata infettata con un worm che sfruttava difetti nei programmi di utility dei sistemi basati su versioni derivate-BSD di UNIX. La falla ha permesso ai programmi di entrare in queste macchine ed auto-replicarsi infettando i sistemi ed interrompendo la normale attività di Internet per molti giorni. Questo attacco rappresenta il primo grande attacco di tipo DoS che la rete ha conosciuto nella sua storia.

Sebbene UNIX sapesse da tempo delle proprie vulnerabilità di sicurezza, la diffusione del problema ha stupito quasi tutti. L’effetto più notevole è stato il fatto che il numero dei processi in esecuzione sui sistemi aumentava sempre più come venivano infettati. Nel tempo alcune di queste macchine divennero incapaci di eseguire qualsiasi cosa. Uno dei primi metodi per arginare la diffusione del worm era rinominare uno o entrambi dei compilatori C per prevenire il loro uso. Inoltre, ogni volta che si scopriva un qualche modo per risolvere il problema, questo veniva pubblicizzato in modo che tutti potessero adottarlo in breve tempo.

**Worm vs Virus**

Un worm è un programma che viene eseguito indipendentemente e può propagare una versione completa e funzionante di sé stesso ad altre macchine.

Un virus è un pezzo di codice che si aggiunge ad altri programmi, incluso il sistema operativo. Non può essere eseguito indipendentemente ma richiede che il suo programma ospite sia eseguito per essere attivato. Include la possibilità di creare una copia evoluta di sé stesso.

**Come opera il worm**

Il Worm ha approfittato di difetti nel software standard installato su molti sistemi UNIX e di un meccanismo utilizzato per semplificare la condivisione delle risorse all'interno di reti locali.

Difetti e modi in cui la sicurezza veniva bucata:

* il programma finger è un utility che permette agli utenti di ottenere informazioni riguardo ad altri utenti (come nome completo utilizzato nel login o se è attualmente connesso); questo programma funziona come un demone o un processo in background che soddisfa richieste remote usando il protocollo finger. Il worm ha sfruttato il fatto che il programma utilizza alcune routine della libreria I/O del C (*gets*, ma ne esistono anche altre) che leggono input senza controllare il limite del buffer coinvolto (buffer overflow). In questo modo l'input che eccede la grandezza del buffer ha sovrascritto lo stack modificando il comportamento del programma
* il programma sandmail è un demone ed è stato progettato per instradare le mail in un sistemi di reti eterogeneo. Il worm ha sfruttato il mailer che opera in background per replicarsi utilizzando un bug che aveva a che fare con le funzionalità fornite da un opzione di debug nel codice
* un attacco chiave del Worm coinvolgeva il tentativo di scoprire le password degli utenti. Ha avuto successo perché la password criptata degli utenti era in un file publicly-readable. Una volta ottenuto il worm ha decriptato le password con un attacco a dizionario reso ancora più semplice dal fatto che gli utenti utilizzavano password comuni. Ottenuto l'accesso al sistema si è replicato
* per replicarsi il worm ha sfruttato anche il fatto che alcuni sistemi permettevano di collegarsi ad una macchina senza autenticarsi ogni volta, perché si era ritenuti trusted. Il worm utilizzava un meccanismo per trovare le macchine che ritenevano il sistema in cui era trusted e poi inviava dei comandi a queste macchine che li eseguivano perché provenienti da una macchina trusted

**Composizione del Worm**

Il Worm consisteva di due parti: un main program e un bootstrap program (99 righe di codice C che venivano compilate sulla macchina da infettare, andando a richiamare il compilatore con cui veniva compilato - soluzione: andare a cambiare il nome del compilatore).

Il main program, una volta stabilito su una macchina, può ottenere informazioni su altre macchine nella rete alla quale la macchina corrente è connessa. Il worm ha sfruttato i 4 difetti del paragrafo superiore per stabilire il suo bootstrap program su quelle macchine.

A seguito di questo ed altri attacchi è stato creato il CERT, ente che si dovrebbe occupare di coordinare le operazioni in situazioni di emergenze informatiche (nel caso del worm per aggiustare la rete cercavano di comunicare attraverso la rete ma la rete era down).

Internet è un algoritmo complesso, e generalmente c'è un problema di omogeneità ed eterogeneità: più eterogenea è la rete più è difficile che un worm riesca ad infettare la rete come è successo nell'88. Esistono delle classi di dispositivi uguali tra loro: console di gioco.

Com'è stato fermato il virus?

## Consigli di lettura 4 - Implementation and Implications of a Stealth Hard-Drive Backdoor - J. Zaddach

I moderni server e workstation si fidano implicitamente che gli hard disk agiscano come dispositivi corretti. Questa lettura analizza la catastrofica perdita di sicurezza quando gli hard disk non sono affidabili. Mostra come è possibile compromettere il firwmare di un hard disk ricorrendo solo a informazioni pubbliche o reverse engineering. Utilizzando tale firmware compromesso, viene presentato un rootkit invisibile che sostituisce blocchi arbitrari dal disco mentre vengono scritti, fornendo una backdoor attraverso la quale si sostituiscono i dati. L’overhead delle performance in un dispositivo compromesso è solo dell’1% rispetto ad uno che funziona correttamente.

Un attaccante può stabilire un canale di comunicazione con il disco compromesso per infiltrare comandi ed estrarre informazioni.

Rootkit e backdoor sono esempi di codice malizioso progettati per rimanere non-rilevati e raccogliere informazioni per un lungo periodo di tempo, permettendo all’attaccante di mantenere il controllo di macchine compromesse.

Dal punto di vista dell’attaccante, uno svantaggio delle backdoor nell’hardware è il fatto che sono molto dipendenti dall’hardware, richiedendo personalizzazioni per ogni dispositivo obiettivo, rendendole meno attrattive delle tradizionali backdoor nei sistemi operativi. Tuttavia nel mercato ci sono solo tre principali produttori, e il supporto da un modello all’altro dello stesso produttore richiede un limitato lavoro, diventando così un vettore di attacco attrattivo.

Spesso vengono usate per compromettere altri componenti del sistema: in questo caso viene fatto bootstrap del codice malizioso sul sistema operativo, così che il malware si propaghi infettando il kernel e usando l’hardware compromesso solo come modo per sopravvivere alla re-installazione e agli aggiornamenti del software.

*Esempio*

Nell’esempio hanno inserito una Data Exfiltration Backdoor (DEB) in un hard disk SATA, che sono i più diffusi in ambito home, utilizzando alcune funzionalità del firmware che sono presenti ad es. a scopo di debug.

La backdoor faceva in modo che ogni volta che il sistema scriveva su disco venisse eseguito del codice aggiuntivo.

Nell'esempio in questione hanno contaminato un hard disk a cui avevano accesso fisico ma hanno constato che sarebbe possibile infettare un hard disk anche da remoto attraverso un Trojan che sovrascrive il firmware utilizzando la possibilità di aggiornare il firmware (il firmware NON è firmato!).

Si è visto come le prestazioni fossero comunque buone nonostante la backdoor ed usando meccanismi come il caching tutte le operazioni aggiuntive eseguite dalla backdoor vengono mascherate all'utente e la percentuale di rallentamento è presente ma è difficile da individuare.

La backdoor consente di leggere e/o estrarre informazioni da un hard disk ed ha bisogno di essere accessibile dall'esterno. Per farlo hanno sfruttato alcune funzionalità del server sul quale è installato l'hard disk, ad es. un forum, che servizio dove gli utenti esterni hanno privilegi di scrittura, e, tramite messaggi sul forum, l'attaccante può inviare richieste alla backdoor che risponde allo stesso modo sul forum.

Per fare questo hanno usato una sorta di *magic number* da inserire all'interno dei messaggi che quando viene rilevato dalla backdoor viene eseguito il codice.

Nel mettere in pratica questo possono esserci dei problemi di codifica, caching (da un lato maschera le prestazioni leggermente degradanti a causa della presenza della backdoor ma dall'altro può causare ritardi nella ricezione del comando dannoso o nella risposta) e di modalità di individuazione dei magic number.

*Come prevenire/difendersi questo attacco*? Più possibilità: una già utilizzata è crittografia dei dati (crittografia hardware - ma la backdoor potrebbe intercettare la chiave di crittografia - o software), una che loro propongono è implementare controlli di corrispondenza tra la cache del sistema operativo e quello che è scritto effettivamente su disco, o implementare un controllo di integrità sul firmware.

Tutto questo si può applicare a qualsiasi hardware che sia dotato di firmware aggiornabile.

**Considerazioni generali relative anche alla lettura 5**

La lettura 5 parla di Internet of Things e spiega che tante delle "cose" che si trovano in rete hanno un firmware che non è aggiornabile.

Da un lato un firmware non aggiornabile non può essere aggiornato e quindi modificato (se c'è una vulnerabilità ci rimane però non puoi aggiungerne di nuove). Nel caso invece di un device avente firmware aggiornabile come un hard disk quello che si ha è un meccanismo in cui è sufficiente essere root per fare aggiornamento del firmware (non c'è una modalità per verificare che versione del firmware c'è in quel momento, non sai se qualcuno ha sostituito quel firmware).

I firmware non sono firmati: non c'è alcun modo per verificare che il firmware sia quello giusto, che non sia stato alterato, non c'è modo per verificare quale firmware hai caricato. Di conseguenza ci fidiamo del nostro hardware.

Esistono soluzioni? Dipende: se non hai sicurezza fisica non puoi avere sicurezza e deve essere possibile che i produttori forniscano un modo per aggiornare i firmware in maniera sicura.

## Consigli di lettura 5 - The Internet of Things Is Wildly Insecure And Often Unpatchable - Wired

Con l'Internet of Things molti dispositivi elettronici sono connessi tra loro. Questi dispositivi sono spesso costruiti con software embedded e non aggiornabile per vari motivi (ad esempio perché utilizzano hardware obsoleto, perché la casa madre non rilascia aggiornamenti, perché gli aggiornamenti non sono previsti, perché il codice sorgente non viene rilasciato e nessuno può modificarlo).

I sistemi embedded sono quindi facilmente attaccabili.

L’articolo prende in considerazione i prodotti commercialmente disponibili ed utilizzati su larga scala e si nota come in ogni casa, ad esempio, ci sia un router che ha le stesse capacità di computazione di un computer degli anni '90. Infatti, nei router commerciali di solito il software è vecchio di 4 anni.

Se una volta l’aggiornamento non era un problema perché i dispositivi erano "standalone" (ovvero chiusi in sé stessi), adesso lo è diventato in quanto i sistemi di computazione sono sempre attivi, inseriti all’interno di qualsiasi strumento (embedded) e soprattutto collegati alla rete, e quindi in grado di comunicare con il resto del mondo (Internet, Bluetooth). Tra essi abbiamo, per esempio, smart TV e altri elettrodomestici intelligenti, ma soprattutto router e switch di rete, che dovrebbero essere aggiornati per mantenere un buon livello di sicurezza.

Il vero focus dell'articolo è che molti di questi dispositivi sono spesso non patchabili.

Soluzione proposta: da un punto di vista pratico le grandi aziende dovrebbero metterci più impegno a fare codice che sia più mantenibile, meno soggetto ad errori; da un punto di vista etico cerca di convincere chi acquista i prodotti a non boicottare questi prodotti ma ad andare verso prodotti più sicuri.

Una soluzione proposta, è che da un punto di vista pratico le grandi aziende dovrebbero impegnarsi a fare codice che sia più sicuro, più mantenibile, meno soggetto ad errori e riusabile. Dal punto di vista etico si cerca di convincere chi acquista i prodotti a non boicottare tali prodotti ma scegliere prodotti più sicuri.

Avere il possesso dei sorgenti non implica che si riesca a patchare il dispositivo di cui si hanno i sorgenti, se non si ha la possibilità di aggiornare il firmware.

## Consigli di lettura 6 - Cookieless Monster- Exploring the Ecosystem of Web-based Device Fingerprinting

*Premesse*: la device fingerprinting permette di identificare utenti e dispositivi di elaborazione remoti anche quando i cookie sono disattivati.

L’abilità di tracciare gli utenti e le loro abitudini online è molto lucrativa, così come intrusiva per la privacy degli utenti.

Il fingerprinting non è sempre usato a scopo negativo. Ad esempio, si può usare per verificare che un utente che sta facendo accesso alla banca con un dispositivo diverso dal solito, sia effettivamente lui o se qualcun altro sta usando la sua identità.

L’articolo ha analizzato il codice di tre popolari fornitori di codici browser-fingerprinting, rivelando le tecniche che permettono ai siti di tracciare gli utenti.

* *Fingerprinting attraverso plugin popolari*: tutte le compagnie usano Flash per l’identificazione a causa delle sue vulnerabilità. Adobe Flash è un plugin proprietario di diffusa adozione nei browser, che nonostante povere performance, assenza di stabilità e nuove tecnologie, è ancora largamente disponibile. Nei moderni browser si hanno estensioni che disabilitano Flash per caricarlo solo nel caso in cui venga esplicitamente richiesto dall’utente.
* *Vendor specific fingerprinting*: le società di fingerprinting non operano allo stesso modo in tutti i browser; quando riconoscono uno specifico browser allora cercano di fare fingerprinting delle specifiche proprietà di quel browser.
* *Lista di font del sistema*: ottenere la lista di font utilizzati dal sistema può servire come parte di un'impronta unica relativa ad un utente. Questa lista non è fornita dal browser direttamente ma si può acquisire utilizzando un plugin che fornisce volontariamente queste informazioni o utilizzando un canale laterale che rivela indirettamente la presenza o l'assenza di qualsiasi dato font.
* *Detection of HTTP proxies*: una delle caratteristiche più difficili da falsificare per un client è il suo indirizzo IP. Per nascondere l'indirizzo IP di un utente viene utilizzato un proxy che fa da relay tra il client (che desidera rimanere nascosto) e una terza. Queste compagnie di fingerprinting hanno visto che se una richiesta proveniente JavaScript contiene lo stesso token contenuto in una richiesta Flash proveniente da un diverso indirizzo IP sorgente originaria, il server può essere certi che l'utente sta utilizzando un proxy HTTP.

Le diversità nell’implementazione dei browser vengano sfruttate per distinguere non solo la famiglia di browser, ma anche versioni maggiori e minori.

In passato veniva incentivato l’uso di plugin ed estensioni del browser che permettevano di nascondere lo user agent ed essere quindi meno rintracciabili. La ricerca condotta all'interno dell'articolo ha evidenziato l'opposto: questa copertura incompleta permette alle compagnie di fingerprinting di rilevare il fatto che l’utente sta cercando di nascondersi, rendendosi così “particolare” e più visibile.

Gli autori dell'articolo ritengono che, data la complessità di nascondere completamente la vera natura di un browser, questo possa essere fatto in modo efficiente solo dai produttori di browser. Indipendentemente dalla loro complessità e raffinatezza, browser plugin ed estensioni non saranno mai in grado di controllare tutto ciò che può controllare un fornitore di browser.

Gli autori dell'articolo, inoltre, hanno discusso di possibili soluzioni per ridurre la superficie fingerprintable di un utente e descritto usi del fingerprinting che possono diventare più prevalenti in futuro.

## Consigli di lettura 7 - A Survey of Defense Mechanisms Against DDoS Service Flooding Attacks

DDoS Flooding Attacks sono una delle maggiori minacce alla sicurezza, e consistono nel tentativo di interrompere il legittimo accesso ai servizi da parte degli utenti. Gli attaccanti guadagnano l’accesso a un grande numero di computer sfruttando le loro vulnerabilità, per creare delle Botnet.

L’attacco è coordinato, distribuito, a grande-scala e rivolto verso uno o più obiettivi. La natura distribuita degli attacchi DDoS li rende difficili da combattere e tracciare a ritroso, perché spesso gli attaccanti usano spoofed IP address per non essere rintracciati.

**Modi per lanciare un attacco DDoS**

1. L’attaccante invia pacchetti malformati alla vittima per confondere il protocollo o un’applicazione in esecuzione.
2. (Più comune) L’attaccante cerca di

* Interrompere la connettività di un utente legittimo esaurendo banda, la capacità del router di processare i pacchetti dei router o le risorse di rete – a livello di rete/trasporto
* Interrompere il servizio di un utente legittimo esaurendo le risorse del server come socket, CPU, memoria, banda) – a livello di applicazione

Le motivazioni alla base di un attacco di tipo DDoS possono essere diverse: economico/finanziarie, vendetta, ideologiche, sfide, cyberwarfare.

**Categorie di DDoS Flooding Attacks**

A seconda del protocol level a cui si rivolgono esistono

* Network/transport-level DDoS flooding attacks, lanciati tramite pacchetti di protocollo TCP, UDP, ICMP e DNS

Esempio di attacco di questo tipo è il *Flooding Attack* che consiste nell'intasare la banda dell'host obiettivo di pacchetti in modo che il servizio attaccato non possa funzionare nei confronti degli utenti legittimi

* Application-level DDoS flooding attacks, consistono nell'interrompere un servizio esaurendo le risorse del server; sono più nascosti perché l'attaccante si espone di meno e consuma meno banda ma hanno lo stesso impatto sui servizi dal momento che hanno come obiettivo caratteristiche specifiche di applicazioni come HTTP, DNS, SIP.

**Botnet**

Gli attacchi sono lanciati da una rete di Zombie o Botnet computer controllati da remoto, che simultaneamente e continuamente inviano un grande ammontare di traffico e/o richieste di servizio ad un sistema obiettivo. Tali computer sono reclutati tramite worms, Trojan horse o backdoor.

Una Botnet è controllata da un attaccante ed è composta da masters (attaccante), handlers e bots.

Gli handler sono mezzi di comunicazione che gli attaccanti usano per comunicare con i bots. Il protocollo più usato dagli attaccanti per comunicare con i bots inviandogli comandi è Internet Relay Chat (IRC).

Le bots sono le macchine che sono state compromesse attraverso gli handlers.

In base al modo in cui le bots sono controllate dall'attaccante le principali categorie in cui le botnet possono essere divise sono botnet

* IRC-based (usa IRC che è un protocollo di messaggistica istantanea on-line, basato su una architettura client/server con canali di default per comunicare tra i server; questi canali vengono utilizzati dagli attaccanti come handler per reclutare e comunicare coi bot e possono facilmente nascondersi a causa del volume di traffico che i server IRC hanno)
* Web-based (usa HTTP come protocollo di comunicazione e i bot scaricano le istruzioni di attacco tramite richieste web)
* P2P-based

Per difendersi da questo tipo di attacchi bisogna aver chiaro dove difendersi e quando difendersi.

Poiché un attacco DDoS spreca molte risorse nel percorso che porta all’obiettivo, i meccanismi di difesa devono rilevarlo il prima possibile e fermarlo il più vicino alla sorgente d’attacco.

Ci sono due criteri per classificare i meccanismi di difesa

* in base alla luogo dove vengono implementati: source-based, destination-based, network-based, hybrid.
* in base al momento in cui agiscono in risposta ad un possibile attacco: prima (prevenzione), durante (rilevazione) e dopo l’attacco (identificazione e risposta).

Obiettivo della lettura è ragionare, quando progettiamo un servizio, se il nostro servizio sia esposto o meno a questi tipi di attacchi.

## Consigli di lettura 8 - Buffer Overflows - Attacks and Defenses for the Vulnerability of the Decade

Buffer Overflow è la più comune forma di vulnerabilità alla sicurezza degli ultimi dieci anni, attraverso la quale un utente cerca di ottenere il controllo totale o parziale di un host. Se le vulnerabilità di buffer overflow potessero essere eliminate in modo efficace, una gran parte delle minacce alla sicurezza più gravi verrebbero anch'esse eliminate.

Tale vulnerabilità permette all’attaccante di inserire e eseguire codice d’attacco, che viene eseguito coi privilegi del programma vulnerabile.

L’obiettivo di un attacco di buffer overflow è quello di sovvertire la funzione di un programma privilegiato in modo che l’attaccante possa prendere il controllo del programma e, se il programma ha sufficienti privilegi, anche il controllo dell’host.

Generalmente si attacca un programma root e si esegue codice per avere una root shell.

Per ottenere l'obiettivo di cui sopra l'attaccante deve prima raggiungere due sotto-obiettivi

* **fare in modo che codice adatto sia disponibile nello spazio degli indirizzi del programma**

Per fare questo si può procedere in due modi

* *iniettando* una stringa come input del programma che viene memorizzata in un buffer (la stringa contiene byte che sono istruzioni CPU native per la piattaforma che vuole attaccare)

L'attaccante non deve necessariamente effettuare un overflow di un buffer; alcuni payload potrebbero benissimo essere iniettati in buffer di dimensioni contenute, adibiti per precisi scopi e funzionalità del programma.

Il buffer può essere localizzato ovunque: sullo stack o sull’heap (variabili allocate dinamicamente).

* *utilizzando codice già presente nello spazio degli indirizzi del programma*; se il codice è già presente l'attaccante non deve fare altro che parametrizzarlo per fare in modo che il programma salti al codice
* **fare in modo che il programma salti a tale codice, alterando il flusso di controllo del programma attraverso appositi parametri caricati nei registri e nella memoria**

Per fare questo l'attaccante può

* *corrompere l'activation record* alterando l’indirizzo di ritorno a cui saltare quando la funzione termina di una funzione vittima a cui il programma deve saltare quando la funzione termina

Questa forma di Buffer Overflow si chiama “Stack smashing attack”.

* *usare function pointer*: i puntatori a funzione possono essere allocati ovunque (stack, heap, static data area); l’attaccante ha solo bisogno di trovare un buffer adiacente ad un puntatore a funzione ed effettuare l’overflow del buffer per modificare facendo in modo che salti alla posizione immessa dall'attaccante
* *longjump buffers*

La tecnica più utilizzata per fare buffer overflow combina un iniezione di codice e la corruzione dell'activation record.

**Approcci per difendersi da buffer overflow**

Quattro sono gli approcci per difendersi da buffer overflow

* scrivere codice corretto,
* rendere le aree di memorizzazione non-eseguibili (non si può rendere *ogni* segmento dati nello spazio degli indirizzi non-eseguibile senza sacrificare sostanzialmente la compatibilità, per cui per preservarla si rende non-eseguibile solo lo stack),
* eseguire un controllo dei confini degli array (bound checking) ad ogni accesso in lettura/scrittura all’array,
* eseguire un controllo d’integrità (integrity checking) sui code pointer prima che siano dereferenziati.

Rispetto al bound checking l'integrity checking non risolve perfettamente il problema di buffer overflow ma porta con sé una serie di vantaggi in termini di performance, compatibilità con il codice esistente e sforzi implementativi.

StackGuard è una tecnica di compilazione che esegue il controllo dell’integrità dei code pointer controllando l’indirizzo di ritorno negli activation record delle funzioni.

PointGuard è una generalizzazione di StackGuard per difendersi da attacchi di buffer overflow più generali.

Il risultato dell’analisi mostra che una strategia di difesa basata su StackGuard e rendere lo stack non-eseguibile sconfigge molti degli attacchi di buffer overflow contemporanei; molti dei restati attacchi vengono sconfitti da una strategia di difesa basata su PointGuard.

.

Quella del Buffer Overflow, è una tecnica che colpisce solitamente programmi scritti in C o in C++. In particolare quei programmi che usano vecchie funzioni per allocare memoria, e che in automatico non effettuano alcun controllo sullo spazio allocato e l’effettiva lunghezza dell’input passato dall’utente. Ad esempio: strcpy, gets, sprintf etc…

Tra le soluzioni più efficaci ad un buffer overflow, oltre a quelle scontate, di utilizzare funzioni che controllino la lunghezza degli input, ve ne sono anche di più drastiche:

– Stack non eseguibile.  
– Modifiche al compilatore

Che però risultano attualmente poco attuabili. Perchè molti programmi standard richiedono l’esecuzione di routine sullo stack, per sistemi di ottimizzazione e quant’altro. Oppure nel secondo caso, modifiche al compilatore minano alle performance del programma.

Il paper conclude dicendo che ci sono alcune tecniche di difesa: primo meccanismo sarebbe fare codice sicuro (strncopy) ma è un utopia, stackguard, NX bit (stack non eseguibile) aslr (cambiare indirizzi di memoria in maniera randomica ad ogni esecuzione) sono bypassabili, non è un problema solo di c/c++ perché i compilatori, gli interpreti.. sono scritti in c.