

```
>total; walk->total -= walk->entrylen; return hash_walk_next(walk);}int crypto_hash_walk_done(struct crypto_hash_walk *walk, int err){ unsigned int alignmask = walk->alignmask; unsigned int nbytes = walk->entrylen; walk->data -= walk->offset; if (nbytes && walk->offset && alignmask && !err) { walk->offset = ALIGN(walk->offset, alignmask + 1); walk->data += walk->offset; nbytes = min(nbytes, ((unsigned int)(PAGE_SIZE)) - walk->offset); walk->entrylen -= nbytes; return nbytes; } if (walk->flags & CRYPTO_ALG_ASYNC) kunmap(walk->pg); else { kunmap_atomic(walk->data); /* The may sleep test only makes sense for sync users. * Async users don't need to sleep here anyway. */ crypto_yield(walk->flags); } if (err) return err; if (nbytes) { walk->offset = 0; walk->pg++; return hash_walk_next(walk); } if (!walk->total) return 0; walk->sg = sg_next(walk->sg); return hash_walk_new_entry(walk);}EXPORT_SYMBOL_GPL(crypto_hash_walk_done);int crypto_hash_walk_first(struct ahash_request *req, struct crypto_hash_walk *walk){ walk->total = req->nbytes; if (!walk->total) { walk->entrylen = 0; return 0; } walk->alignmask = crypto_ahash_alignmask(crypto_ahash_reqtfm(req)); walk->sg = req->src; walk->flags = req->base.flags & CRYPTO_TFM_REQ_MASK; return hash_walk_new_entry(walk);}EXPORT_SYMBOL_GPL(crypto_hash_walk_first);int crypto_ahash_walk_first(struct ahash_request *req, struct crypto_hash_walk *walk){ walk->total = req->nbytes; if (!walk->total) { walk->entrylen = 0; return 0; } walk->alignmask = crypto_ahash_alignmask(crypto_ahash_reqtfm(req)); walk->sg = req->src; walk->flags = req->base.flags & CRYPTO_TFM_REQ_MASK; walk->flags |= CRYPTO_ALG_ASYNC; BUILD_BUG_ON(CRYPTO_TFM_REQ_MASK & CRYPTO_ALG_ASYNC); return hash_walk_new_entry(walk);}EXPORT_SYMBOL_GPL(crypto_ahash_walk_first);static int ahash_setkey_unaligned(struct crypto_ahash *tfm, const u8 *key, unsigned int keylen){ unsigned long alignmask = crypto_ahash_alignmask(tfm); int ret; u8 *buffer, *alignbuffer; unsigned long absize; absize = keylen + alignmask; buffer = kmalloc(absize, GFP_KERNEL); if (!buffer) return -ENOMEM; alignbuffer = (u8 *)ALIGN((unsigned long)buffer, alignmask + 1); memcpy(alignbuffer, key, keylen); ret = tfm->setkey(tfm, alignbuffer, keylen); kfree(buffer); return ret;}int crypto_ahash_setkey(struct crypto_ahash *tfm, const u8 *key, unsigned int keylen){ unsigned long alignmask = crypto_ahash_alignmask(tfm); if ((unsigned long)key & alignmask) return ahash_setkey_unaligned(tfm, key, keylen); return tfm->setkey(tfm, key, keylen);}EXPORT_SYMBOL_GPL(crypto_ahash_setkey);static int ahash_nosetkey(struct crypto_ahash *tfm, const u8 *key, unsigned int keylen){ return -ENOSYS;}static inline unsigned int ahash_align_buffer_size(unsigned len, unsigned long mask){ return len + (mask & ~(crypto_tfm_ctx_alignment() - 1));}static int ahash_save_req(struct ahash_request *req, crypto_completion_t cplt){ struct crypto_ahash *tfm = crypto_ahash_reqtfm(req); unsigned long alignmask = crypto_ahash_alignmask(tfm); unsigned int ds = crypto_ahash_digestsize(tfm); struct ahash_request_priv *priv; priv = kmalloc(sizeof(*priv) + ahash_align_buffer_size(ds, alignmask), GFP_KERNEL); if (!priv) return -ENOMEM; /* WARNING: Voodoo programming below! * The code below is obscure and hard to understand, thus explaining it is not the goal of this document. * To understand the layout of structures used here! * The code here will replace the original request with a new one and buffers so the hashing operation can store the result in aligned buffer. We will call the modified request "priv" and the original request "req". The new request will look as such: * req { .result = ADJUSTED[new aligned buffer] * .base.complete = ADJUSTED[pointer to completion function] * .base.data = ADJUSTED[*req (pointer to self)] * .priv = ADJUSTED[new priv] { * .result = ORIGINAL(result) * .complete = ORIGINAL(base.complete) * .data = ORIGINAL(base.data) * } } */ priv->result = req->result; priv->complete = req->base.complete; priv->data = req->base.data; /* WARNING: We do not backup req->priv here! The result is stored in the new request. * user must NOT_EVER depend on it's content! */ req->result = PTR_ALIGN((u8 *)priv->ubuf, alignmask + 1); req->base.complete = cplt; req->base.data = req; req->priv = priv; return 0;}static void ahash_restore_req(struct ahash_request *req){ struct ahash_request_priv *priv = req->priv; /* Restore the original crypto request. */ req->result = priv->result; req->base.complete = priv->complete; req->base.data = priv->data; req->priv = NULL; /* Free the req->priv.priv from the ADJUSTED request. */ kfree(priv);}static void ahash_op_unaligned_finish(struct ahash_request *req, int err){ struct ahash_request_priv *priv = req->priv; if (err == -EINPROGRESS) return; if (!err) memcpy(priv->result, req->result, crypto_ahash_digestsize(crypto_ahash_reqtfm(req))); ahash_restore_req(req);}static void ahash_op_unaligned_done(struct crypto_async_request *req, int err){ struct ahash_request *areq = req->data; /* Restore the original request, see ahash_op_unaligned() for what * goes where. * The "struct ahash_request *req" here is in fact the "req.base" * from the ADJUSTED request from ahash_op_unaligned(), thus as it * is a pointer to self, it is also the ADJUSTED "req". */ /* First copy req->result into req->priv.result */ ahash_op_unaligned_finish(areq, err); /* Complete the ORIGINAL request. */ areq->base.complete(&areq->base, err);}static int ahash_op_unaligned(struct ahash_request *req, int (*op)(struct ahash_request *)){ int err; err = ahash_save_req(req, ahash_op_unaligned_done); if (err) return err;
```

Ransomware

COSA SONO - COME DIFENDERSI

Malware

Software creato per uno scopi malevoli:

Rubare informazioni

Rendere la macchina infetta veicolo di traffico malevolo

(mail bombing, DoS, Botnet ...)

Mostrare pubblicità indesiderata

...

I più comuni veicoli di malware sono:

- Virus

- Trojan Horses

- Worms

```
>total; walk->total -= walk->entrylen; return hash_walk_next(walk);}int crypto_hash_walk_done(struct crypto_hash_walk *walk, int err){ unsigned int alignmask = walk->alignmask; unsigned int nbytes = walk->entrylen; walk->offset += walk->entrylen; if (nbytes && walk->offset & alignmask && !err) { walk->offset = ALIGN_MASK(walk->offset, alignmask + 1); walk->data += walk->offset; nbytes = min(nbytes, ((unsigned int)(PAGE_SIZE)) - walk->offset); walk->entrylen -= nbytes; return nbytes; } if (walk->flags & CRYPTO_ALG_ASYNC) kunmap(walk->pg); else { kunmap_atomic(walk->data); /* The may sleep test only makes sense for sync users. * Async users don't need to sleep here anyway. */ crypto_yield(walk->flags); } if (err) return err; if (nbytes) { walk->offset = 0; walk->pg++; return walk->total - nbytes; } if (!walk->total) return 0; walk->sg = sg_next(walk->sg); return hash_walk_new_entry(walk);}EXPORT_SYMBOL_GPL(crypto_hash_walk_first);static struct crypto_hash_walk *crypto_hash_walk(struct crypto_hash_walk *walk){ walk->total = walk->entrylen; if (!walk->total) return 0; walk->alignmask = crypto_alignmask(crypto_hash_reqtfm(walk->req)); walk->sg = req->sg; walk->flags = req->base.flags & CRYPTO_TFM_REQ_MASK; return hash_walk_new_entry(walk);}EXPORT_SYMBOL_GPL(crypto_hash_walk_first);static struct crypto_hash_walk *crypto_hash_walk(struct crypto_hash_walk *walk){ walk->total = req->nbytes; if (!walk->total) { walk->entrylen = 0; return 0; } walk->alignmask = crypto_alignmask(crypto_hash_reqtfm(walk->req)); walk->sg = req->sg; walk->flags = req->base.flags & CRYPTO_TFM_REQ_MASK & CRYPTO_ALG_ASYNC; return hash_walk_new_entry(walk);}EXPORT_SYMBOL_GPL(crypto_hash_walk_first);static int ahash_setkey_unaligned(struct crypto_ahash *tfm, const u8 *key, unsigned int keylen){ unsigned long alignmask; int ret; u8 *buffer, *alignbuffer; unsigned long absize; absize = keylen + alignmask; buffer = kmalloc(absize, GFP_KERNEL); if (!buffer) return -ENOMEM; alignbuffer = (u8 *)ALIGN(0, alignmask + 1); memcpy(alignbuffer, key, keylen); ret = tfm->setkey(tfm, alignbuffer, keylen); kfree(buffer); return ret;}EXPORT_SYMBOL_GPL(crypto_ahash_setkey);static int ahash_setkey(struct crypto_ahash *tfm, const u8 *key, unsigned int keylen){ return tfm->setkey(tfm, key, keylen);}EXPORT_SYMBOL_GPL(crypto_ahash_setkey);static int ahash_nosetkey(struct crypto_ahash *tfm, const u8 *key, unsigned int keylen){ return 0;}EXPORT_SYMBOL_GPL(crypto_ahash_nosetkey);static int ahash_save_req(struct ahash_request *req, crypto_completion_t cplt){ struct crypto_ahash *tfm = crypto_ahash_reqtfm(req); unsigned int ds = crypto_ahash_digestsize(tfm); struct ahash_request_priv *priv; priv = kmalloc(sizeof(*priv), GFP_KERNEL); if (!priv) return -ENOMEM; /* WARNING: Voodoo programming below! * The code below is obscure and hard to understand, thus explanation * is necessary. See include/crypto/hash.h and include/linux/crypto.h * to understand the layout of structures used here! * The code here will replace portions of the ORIGINAL request with * pointers to new code and buffers so the hashing operation can store * the result in aligned buffer. We will call the modified request * an ADJUSTED request. * The newly mangled request will look as such: * req { .result = ADJUSTED[new aligned buffer] .base.complete = ADJUSTED[pointer to completion function] .base.data = ADJUSTED[*req (pointer to self)] .priv = ADJUSTED[new priv] } */ priv->complete = ORIGINAL(base.complete); priv->data = ORIGINAL(base.data); priv->result = ORIGINAL(req->result); priv->priv = req->priv; /* user must NOT_EVER depend on it's content! */ req->result = PTR_ALIGN(priv->result, alignmask + 1); req->base.complete = cplt; req->base.data = req; req->priv = priv; return 0;}static void ahash_restore_req(struct ahash_request_priv *priv){ struct ahash_request_priv *priv = req->priv; /* Restore the original crypto request. */ req->result = priv->result; req->base.complete = priv->complete; req->base.data = priv->data; req->priv = NULL; /* Free the req->priv.priv from the ADJUSTED request. */ kfree(priv->priv);}static void ahash_op_unaligned(struct ahash_request *req, int err){ struct ahash_request_priv *priv = req->priv; if (err == -EINPROGRESS) return; if (!err) memcpy(priv->result, req->result, crypto_ahash_digestsize(crypto_ahash_reqtfm(req))); ahash_restore_req(req);}static void ahash_op_unaligned(struct crypto_async_request *req, int err){ struct ahash_request *areq = req->data; /* Restore the original request, see ahash_op_unaligned() for what * goes where. * The "struct ahash_request *req" here is in fact the "req.base" * from the ADJUSTED request from ahash_op_unaligned(), thus as it * is a pointer to self, it is also the ADJUSTED "req" . */ /* First copy req->result into req->priv.result */ ahash_op_unaligned_finish(areq, err); /* Complete the ORIGINAL request. */ areq->base.complete(&areq->base, err);}static int ahash_op_unaligned(struct ahash_request *req, int (*op)(struct ahash_request *)){ int err; err = ahash_save_req(req, ahash_op_unaligned done); if (err) return err;
```

Come cascarci

Gli utenti vengono indotti nell'eseguire il **malware** utilizzando tecniche di **Ingegneria Sociale**

Per esempio può convincere l'utente

- pretendendo di essere qualcun altro come per esempio

le poste, amazon, ebay o addirittura le forze dell'ordine, oppure mascherandosi da aggiornamento di sistema

- pretendendo di fare del bene (carità, donazioni, ...)

- Millantando guadagni sfruttando l'avidità dell'utente

- Inducendo una paura (il tuo computer è infetto, usa questo tool per ripulirlo)

ECC.:


```
>total; walk->total -= walk->entrylen; return hash_walk_next(walk);}int crypto_hash_walk_done(struct crypto_hash_walk *walk, int err){ unsigned int alignmask = walk->alignmask; unsigned int nbytes = walk->entrylen; walk->data -= walk->offset; if (nbytes && walk->offset & alignmask && !err){ walk->offset = ALIGN(GN(walk->offset, alignmask + 1); walk->data += walk->offset; nbytes = min(nbytes, ((unsigned int)(PAGE_SIZE)) - walk->offset); walk->entrylen -= nbytes; return nbytes; } if (walk->flags & CRYPTO_ALG_ASYNC) kunmap(walk->pg); else { kunmap_atomic(walk->data); /* * The may sleep test only makes sense for sync users. * Async users don't need to sleep here anyway. */ crypto_yield(walk->flags); } if (err) return err; if (nbytes) { walk->offset = 0; walk->pg++; return hash_walk_next(walk); } if (!walk->total) return 0; walk->sg = sg_next(walk->sg); return hash_walk_new_entry(walk);}EXPORT_SYMBOL_GPL(crypto_hash_walk_done);int crypto_hash_walk_first(struct ahash_request *req, struct crypto_hash_walk *walk){ walk->total = req->nbytes; if (!walk->total) { walk->entrylen = 0; return 0; } walk->alignmask = crypto_ahash_alignmask(crypto_ahash_reqtfm(req)); walk->sg = req->src; walk->flags = req->base.flags & CRYPTO_TFM_REQ_MASK; return hash_walk_new_entry(walk);}EXPORT_SYMBOL_GPL(crypto_hash_walk_first);int crypto_ahash_walk_first(struct ahash_request *req, struct crypto_hash_walk *walk){ walk->entrylen = 0; return 0; } walk->alignmask = crypto_ahash_alignmask(crypto_ahash_reqtfm(req)); walk->flags = req->base.flags & CRYPTO_TFM_REQ_MASK; walk->flags |= CRYPTO_ALG_ASYNC; BUILD_BUG_ON(CRYPTO_TFM_REQ_MASK & CRYPTO_ALG_ASYNC); EXPORT_SYMBOL_GPL(crypto_ahash_walk_first);static int ahash_setkey_unaligned(struct crypto_ahash *tfm, const unsigned long alignmask, const unsigned long alignmask(tfm); int ret; u8 *buffer, *alignbuffer; unsigned long absize, GFP_KERNEL); if (!buffer) return -ENOMEM; alignbuffer = (u8 *)ALIGN((unsigned long)absize, alignmask); ret = tfm->setkey(tfm, alignbuffer, keylen); kzfree(buffer); return ret;}int crypto_ahash_setkey_unaligned(struct crypto_ahash *tfm, const unsigned int keylen){ unsigned long alignmask = crypto_ahash_alignmask(tfm); if ((unsigned long)key & alignmask) return -EINVAL; return tfm->setkey(tfm, key, keylen);}EXPORT_SYMBOL_GPL(crypto_ahash_setkey_unaligned);static inline unsigned int ahash_align_buffer_size(unsigned len, unsigned long mask){ return len + (mask & ~(crypto_tfm_ctx_alignment() - 1));}static int ahash_save_req(struct ahash_request *req, struct ahash_request_priv *priv, unsigned long alignmask = crypto_ahash_alignmask(tfm); unsigned int ds = crypto_ahash_digestsize(tfm); struct ahash_request_priv *priv; priv = kmalloc(sizeof(*priv) + ahash_align_buffer_size(ds, alignmask), GFP_KERNEL : GFP_ATOMIC); if (!priv) return -ENOMEM; /* * WARNING: Voodoo programming below! * * The code below is obscure and hard to understand, thus explanation * is necessary. See include/crypto/hash.h and include/linux/crypto.h * to understand the layout of structures used here! * * The code here will replace portions of the ORIGINAL request with * pointers to new code and buffers so the hashing operation can store * the result in aligned buffer. We will call the modified request * an ADJUSTED request. * * The newly mangled request will look as follows: * * .base.complete = ADJUSTED[pointer to completion function] * .base.data = ADJUSTED[new priv] * .result = ORIGINAL(result) * .complete = ORIGINAL(base.data) * } */ priv->result = req->result; priv->complete = req->base.complete; priv->data = req->data; /* Restore the original request */ priv->req = req; /* Restore the original request */ priv->result = req->result; /* Restore the original request */ kzfree(priv->result); return -EINPROGRESS; } static void ahash_restore_req(struct ahash_request *req, struct ahash_request_priv *priv){ /* Restore the original request, see ahash_restore_req() for what * goes where. * * The "struct ahash_request *req" here is in fact the original request, see ahash_restore_req() for what * goes where. * * The "struct ahash_request *req" here is in fact the original request, see ahash_restore_req() for what * goes where. * * The "struct ahash_request *req" here is in fact the original request, see ahash_restore_req() for what * goes where. */ /* First complete the ORIGINAL request. */ areq->base.complete(&areq->base, err);}static int (*op)(struct ahash_request *){ int err; err = ahash_save_req(req, ahash_op_unaligned done); if (err) return err;
```

Ieri:

Lo scopo principale del malware era l'infezione in sé.

Oggi:

Lo scopo principale del malware è quello di generare un profitto

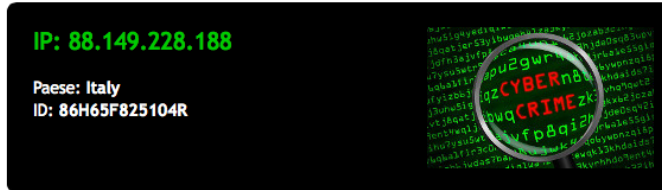


Ransomware

Particolare tipo di malware il cui scopo è quello di indurre chi è colpito al pagamento di un riscatto (ransom).

Prime versioni di questi software cercavano di impedire il funzionamento del computer colpito.

Spesso cercando di ricondurre l'accaduto ad un presunto reato commesso dall'utente promettendo redenzione attraverso il pagamento di una multa.



ATTENZIONE! Il Suo computer personale è stato bloccato per motivi di sicurezza per le seguenti ragioni. Tutte le azioni fatte su questo computer personale sono state registrate. Tutti i vostri file sono stati criptati.

Lei è accusato di visualizzazione / memorizzazione e / o distribuzione di materiale pornografico di contenuto proibito (pornografia infantile / bestialità / stupro, ecc.) Lei ha violato la Dichiarazione Universale sulla lotta contro la diffusione di materiale pedopornografico e accusato di un reato ai sensi dell'articolo 161 del Codice Penale della Repubblica Italiana.

L'articolo 161 del Codice Penale della Repubblica Italiana prevede come pena la privazione della libertà da 5 a 11 anni.

Inoltre, Lei è sospettato di aver violato la "Legge sul diritto d'autore e diritti connessi" (scaricamento della musica pirata, video, software senza licenza), e l'uso e/o la distribuzione di contenuti protetti dai diritti d'autore. In questo modo Lei è sospettato di aver violato l'articolo 148 del Codice Penale della Repubblica Italiana.

L'articolo 148 del Codice Penale della Repubblica Italiana prevede come pena una multa di 150-550 unità di base oppure la privazione della libertà da 3 a 7 anni.

Dal Suo computer personale è stato commesso un accesso non autorizzato all'informazione chiusa all'accesso pubblico ed all'informazione d'importanza statale nella rete Internet.

Lei poteva organizzare l'accesso non autorizzato deliberatamente per motivi di lucro, oppure l'accesso non autorizzato poteva avvenire senza la Sua conoscenza o il Suo consenso, siccome il Suo computer personale potrebbe essere infettato da software maligno. Così, Lei è sospettato, finché l'investigazione sarà condotta, di indagare in violazione involontaria dell'articolo 215 del Codice Penale della Repubblica Italiana ("La legge sull'uso sconsiderato e negligente di computer/PC").

L'articolo 215 del Codice Penale della Repubblica Italiana prevede come pena una multa di €100.000 e/o la privazione della libertà da 5 a 8 anni.

Anche nel corso dell'analisi delle informazioni memorizzate sul Suo computer personale, si è constatato che dal Suo PC in maniera regolare avvengono i mailing di spam di massa, i quali Lei ha volontariamente organizzato per motivi di lucro oppure i quali avvengono senza la Sua conoscenza o il Suo consenso, siccome il Suo computer personale potrebbe essere infettato da software maligno. Questi mailing distribuiscono software dannoso o

Il tempo rimanente: **23:58:47**



Codice PIN Valore

Digitare il codice 100

1 2 3 4 5 6 7 8 9 0 Clear

Pagare PaySafeCard Pagare Ukash

Dove posso acquistare il voucher PaySafeCard?

Paysafecard è disponibile in tutta sicurezza vicino a te in Italia, ad esempio presso numerose edicole, bar, tabaccai anche nei negozi Sisal e Penny. **Panoramica rivenditori:** Smatch, SisalPay, ePay, PayMat, Bitcard, Tigros, Penny, Metro, Marco Polo expert, Iper la Grande i, il gigante, Esselunga, Darty. Shop online: www.WertKartenVerkauf.com



Dove posso acquistare il voucher Ukash?

Puoi richiedere e ottenere Ukash press migliaia di punti vendita, edicole, stazioni di servizio, distributori di benzina, numerosi supermercati, bar e tabacchi e negozi di telefonia mobile dotati di terminale ePay, EpiPoll.



Cryptolocker

Chiamiamo così (è in realtà il nome della variante più comune di questo malware) un particolare tipo di ransomware il cui effetto è quello di criptare i files del computer infettato rendendoli inaccessibili.

Mediante il pagamento del riscatto viene offerta la chiave di decriptazione.

In molte varianti di questo malware non è possibile decifrare i dati con altri metodi



Criptolocker: come funziona un attacco



L'utente viene indotto ad eseguire un software, il metodo più comune è attraverso un **email** che invita a scaricare un documento (tipicamente una **bolletta** o un documento di **spedizione**)

Non è l'unico veicolo, un altro metodo comune è il **malwaretising**, cioè pubblicità ingannevoli iniettate in siti legittimi dove l'utente viene per esempio invogliato a scaricare un coupon.



Il vostro pacchetto con il codice di spedizione **86148195** è arrivato al **23 febbraio 2015**. Corriere non ha espresso un pacco per te. Stampare l'etichetta di spedizione e mostrarlo in ufficio postale più vicino per ottenere il pacchetto.

[Scarica etichetta di spedizione](#)

Se il pacco non viene ricevuto entro 30 giorni lavorativi Sda Express ha il diritto di chiedere un risarcimento da voi per esso sta tenendo nella quantità di 6,14 EUR per ogni giorno di conservazione. È possibile trovare le informazioni sulla procedura e le condizioni di pacchi tenendo l'ufficio più vicino.

Tutela della Privacy

Ci impegniamo a non vendere né condividere in altro modo le informazioni personali che La riguardano, se non nei modi descritti nella presente Informativa sulla protezione dei dati personali. Per erogare i nostri servizi di ritiro e consegna, condividiamo le informazioni con terze parti quali spedizionieri, depositari, paganti terzi e destinatari. Potremo inoltre condividere le informazioni personali che otteniamo con società affiliate, concessionarie, rivenditori e partner commerciali, e queste entità che definiamo "Partner Commerciali SDA" potrebbero usare le informazioni per gli scopi indicati nella presente Informativa sulla protezione dei dati personali. Potremo condividere dati sulla sede fisica con i Partner Commerciali SDA di cui sopra e con altre terze parti ad esempio per il miglioramento dei servizi in base alla posizione (LBS) e lo sviluppo di mappe precise e aggiornate. Inoltre, a meno che Lei non sia contrario, potremo condividere altre informazioni personali con terze parti che non sono Partner Commerciali SDA per scopi quali prodotti o servizi di queste terze parti che potrebbero interessarLe. Le informazioni raccolte da app, strumenti, widget e plug-in di terze parti (come le informazioni raccolte da servizi di accesso a terzi o legate all'uso del tasto "Mi piace" su Facebook) sono raccolte direttamente dai fornitori di queste funzioni. Tali informazioni sono soggette alle informative sulla protezione dei dati personali dei fornitori delle funzioni, e SDA non è responsabile per le procedure di informazione di questi fornitori. [Clicca qui](#) per cancellarli.

Cryptolocker: Anatomia di un attacco



Command and Control C&C

RSA Key exchange

Domain Generation Algorithm

il malware genera un grandissimo numero di contatti verso domini fittizi in modo da camuffare il reale contatto con la rete C&C



In questo momento il malware inizia a criptare i dati

Operazione molto veloce ed eseguita in maniera sempre più intelligente

Lo scopo è quello di rendersi invisibile per il maggior tempo possibile.

Per esempio cercando di criptare per ultimi i files più vicini all'utente.

Vengono disabilitate le shadow copies, criptati per primi i files di rete, delle periferiche collegate al computer ed infine i files dell'utente.

Nuovi tipi di cryptovirus sono particolarmente abili nel non saturare le risorse del computer, non criptare files contigui nello stesso tempo e inserire operazioni diverse tra le operazioni di cifratura per cercare di ingannare antivirus e programmi antimalware specifici per i cryptovirus.

Molte varianti di cryptolocker includono il trojan ZeuS, famoso dal 2007.

Si tratta di un malware in grado di recuperare dati bancari dai form internet.

Ransomware: difese di base

La prima linea di difesa per qualsiasi minaccia informatica è sicuramente:

IL BACKUP

Che deve soddisfare alcune caratteristiche:

- Essere inaccessibile ai sistemi vulnerabili
- Avere un versioning adeguato e un retention consona ai dati protetti
- Deve essere veloce da ripristinare
- Seguire il principio 3-2-1 (almeno 3 copie - in 2 formati differenti - 1 delle quali offsite)

Ransomware: le difese di base

- Diffidenza

- Utilizzo dei sistemi di protezione di base del pc:

- Patch di sicurezza

- Antivirus aggiornato

- Windows smartscreen, UAC, altri software Application Control

- Snapshot dei dischi di rete

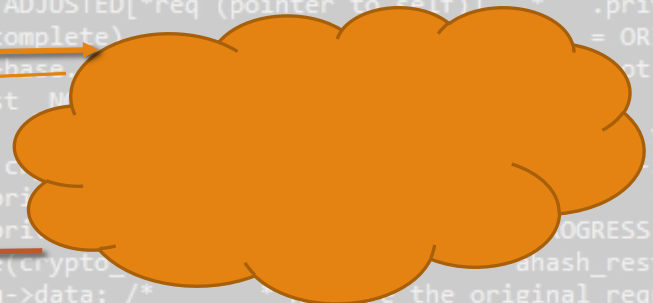
- Software restriction policy (cioè impedire tramite policy aziendale che il computer possa eseguire software in determinate posizioni come l' %APPDATA% dell'utente, le cartelle temporanee di programmi zip, rar, arj ecc..)

Ransomware: difese avanzate LA RETE

L'approccio più seguito oggi è una protezione sempre più avanzata attraverso il primo dispositivo di sicurezza di qualsiasi rete:

Il firewall

Tradizionalmente il firewall viene chiamato stateful, cioè è un apparato capace di permettere le comunicazioni di rete solo se sono «iniziate» dall'interno del perimetro protetto, bloccando le comunicazioni che partono dal lato non protetto della rete (internet per esempio)



Ransomware: difese avanzate LA RETE

Oggi sono sempre più diffusi firewall Layer 7 cioè apparsi in grado di analizzare il flusso delle comunicazioni permesse e decidere se siano o meno pericolose.

Si tratta di dispositivi avanzati in grado per esempio di:

- Rilevare un eventuale malware prima che entri nella rete e quindi bloccarlo prima che raggiunga il computer che lo ha richiesto
- Bloccare l'accesso a determinate reti (per esempio la rete C&C che per lo più si trova su rete anonima Tor)
- Bloccare l'accesso o il download di files in base alla geolocalizzazione della richiesta
- Individuare pattern di comunicazione e confrontarli con pattern noti (intrusion detection)
- Prevenire tentativi di attacco o intrusione (intrusion prevention)
- DNS sinkholes

Oggi le comunicazioni con molti siti, anche quelli che veicolano malware, avvengono con protocollo https.

Un firewall moderno deve essere in grado di analizzare anche queste comunicazioni decifrandole in tempo reale e poi ri-cifrandole verso il client. Si tratta di una funzione complessa detta ssl-inspection, spesso anche non considerata ortodossa perché di fatto è analoga ad un attacco «man in the middle».

Ransomware domani

RaaS sono kit che possono essere acquistati per creare il proprio malware, sfruttato una botnet già pronta e molto potente, è sempre più facile creare la propria estorsione

SamSam un particolare tipo di attacco verso sistemi distribuiti, si cerca di attaccare direttamente i server piuttosto che gli utenti, questo è il primo ransomware conosciuto che sfrutta tecniche di autopropagazione

Furto di identità

LUCA DE FAZIO

E-MAIL: luca@de-fazio.net

CELLULARE: 3388648383