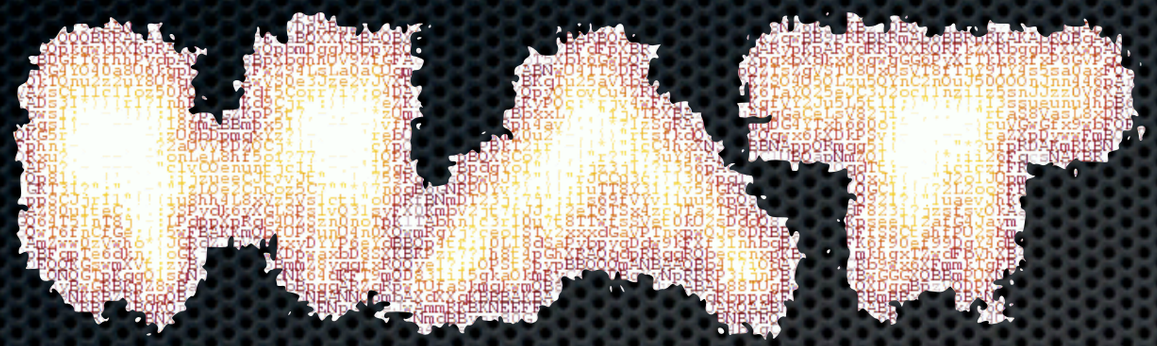


Fuzzing Techniques for Software Vulnerability Discovery

Fuzzing is easy, until you really try it

Relatore: Alberto Trivero

Vulnerability Discovery Methodologies

- White-box testing (source code static analysis)
  - Elevato code coverage ma falsi positivi molto numerosi e necessita dei sorgenti dell'applicazione
- Black-box testing (fuzzing)
  - Sempre applicabile e spesso semplice ma risulta difficile capire quanto del software è stato testato
- Gray-box testing
  - Black-box testing migliorato con tecniche di RCE
- Molti bug hunter intuiscono dove potrebbero annidarsi possibili vulnerabilità: entrano nella testa dello sviluppatore

Fuzz testing, la nascita

- La nascita si fa risalire a Barton Miller che nel 1990 pubblicò “An Empirical Study of the Reliability of UNIX Utilities” presso la University of Wisconsin-Madison
- Il programma sviluppato, *fuzz*, generava un flusso di dati casuali indirizzato a utility UNIX a riga di comando
- Risultato: dal 25% al 33% delle utility andò in crash o si bloccò, a seconda della variante UNIX
- *Il bambino promette bene..* ;-)

Panoramica introduttiva

"Even the most rudimentary fuzzers are surprisingly effective." (Pedram Amini)

- Con automatismi vari, un fuzzer ha lo scopo di identificare degli input che lo sviluppatore non ha valutato e che l'applicazione non è in grado di gestire correttamente
- Il fuzzing è piuttosto diffuso tra i security researchers per la sua relativa semplicità e gli ottimi risultati che offre
- Oltre il 70% delle vulnerabilità che Microsoft ha patchato nel 2006 sono state trovate grazie al fuzzing (esempi passati: IFRAME bug e .printer bug)
- I fuzzer sono responsabili di molti dei "month of" bugs

Giusto qualche nome...

- B0ffuzzer.txt
- COMRaider-0.0.133.exe
- DOM-Hanoi 0.2.html
- EFS-PaiMei.zip
- FileFuzz.zip
- Fuzzled-1.2.tar.gz
- Fuzzware v1.4.rar
- FuzzyFiles.pl
- FuzzySniffAndSend.zip
- GPF.tar.bz2.tar
- JBroFuzz-0.9.exe
- JavaFuzz-0.5.jar
- Malybuzz-1.0b.tar.gz
- PROTOS Test-Suite.zip
- Peach-2.1_BETA3.exe
- RIOT and FaultMon.zip
- SMUDGE.zip
- SPIKE 2.9.tgz
- SPIKEfile.tgz
- Sulley Fuzzing Framework 1.0.exe
- TagBruteForcer.zip
- antiparser-2.0.tar.gz
- autodafe-0.1.tar.gz
- axman-1.0.0.zip
- beSTORM 3.5.6.zip
- bugger-0.01b.tgz
- bunny-0.92.tgz
- chunked fuzzer 0.7.c
- dfuz_0.3.1b.tar.gz
- dhcpfuzz.pl
- fileh.zip
- filep.zip
- ftpfuzz.zip
- fusil-0.8.tar.gz
- fuzzball2-0.7.tar.gz
- hamachi 0.4.html
- iCalfuzz-0.1.py
- ioctlizer-0.1b.zip
- ip6sic-0.1.tar.gz
- ircfuzz-0.3.c
- isic-0.06.tgz
- jsfunfuzz.zip
- mangle.c
- mangleme.tgz
- mielietools-1.0.tgz
- mistress-0.2.rar
- pgmfuzz.c
- powerfuzzer-0.9a.zip
- proxyfuzz-0.1.py
- radius-fuzzer.tgz
- rfuzz-0.9.gem.tar
- scratch.rar
- sfuzz-0.2.c
- smtpfuzz-0.9.16.zip
- snmp-fuzzer-0.1.1.tar.bz2
- sysfuzz.c
- syslog-fuzzer-0.1.pl
- taof-0.3.2.zip
- tftpfuzz.py
- ufuz3.zip
- untidy-beta2.tgz
- zzuf-0.12.tar.gz.tar

Fuzzing phases: un po’ di formalismo

- Identificare l’obiettivo
- Identificare gli input
- Generare degli input malevoli (*fuzzed data*)
- Esecuzione degli stessi
- Monitorare eventuali eccezioni
- Determinare l’exploittabilità

Tipologie di fuzzer

"For each vulnerability that comes out, make sure your fuzzer can find it, then abstract it a bit more." (Dave Aitel)

- Command-line (clfuzz, iFUZZ)
- Environment variabile (sharefuzz, iFUZZ)
- File format (FileFuzz, notSPIKEfile, SPIKEfile)
- Network Protocol (SPIKE, Peach e vari protocol-specific)
- Web application (WebScarab, SPIKE Proxy, wfuzz)
- Web browser (mangleme, DOM-Hanoi, Hamachi, CSSDIE)
- Fuzzing Framework (dfuz, Autodafé, Peach, GPF, Sulley)

Fuzzed data generation

- HTTP GET request standard:
  - GET /index.html HTTP/1.1
- Richieste malformate:
  - AAAAAA...AAAA /index.html HTTP/1.1
  - GET ///////index.html HTTP/1.1
  - GET %n%n%n%n%n%n.html HTTP/1.1
  - GET /AAAAAAAAAAAAAAA.html HTTP/1.1
  - GET /index.html HTTTTTTTTTTTTTTP/1.1
  - GET /index.html HTTP/1.1.1.1.1.1.1.1.1

Fuzzed data generation: Mutation-based

- E’ il metodo più semplice e veloce (da codare e usare)
- Conoscenza scarsa o assente della struttura degli input
- Corruzione (random o con una certa euristica) di input/dati validi
- E’ facile che fallisca in protocolli che utilizzano checksum, CRC, codifiche, bit indicanti la lunghezza di un campo
- `while [1]; do cat /dev/urandom | nc -vv target port; done`
- Esempi: Taof, GPF, FileFuzz, notSPIKEfile

Fuzzed data generation: Generation-based

"Intelligent fuzzing usually gives more results." (Ilja van Sprundel)

- Generazione automatica degli input, senza utilizzarne di precedentemente forniti, basandosi sulle specifiche del formato in esame
- Ogni punto possibile dell'input sarà soggetto a anomalie
- Esempi: SPIKE, SPIKEfile, Sulley e molti dei fuzzer commerciali

Fuzzed data generation: Evolutionary

- Sviluppo ancora in stato embrionale, basato soprattutto sui lavori di Jared DeMott
- Input futuri generati in base alle risposte passate che il programma ha fornito (p.es. basare la priorità dei test cases in base a quale input a raggiunto API pericolose)
- Esempi: Autodafè, EFS (usa PaiMei per tecniche avanzate di code coverage)

Un po’ di euristica

- C’è spesso una ricorrenza nella tipologia di input malevoli in grado di evidenziare una vulnerabilità
- Conviene quindi testare loro prima di fare modifiche random: più completa sarà la lista di input, migliore risulterà il code coverage ottenuto
- Esempi:
  - “A” x 65000
  - ../../../../../../../../../etc/passwd
  - %n%n%n%n%n%n
  - 0xFFFFFFF1

DEMO

Problematiche

- I mutation-based fuzzers possono generare un infinito numero di test cases, quando è stato eseguito a sufficienza il fuzzer?
- Ricerca dei bug non deterministica: stessa probabilità che il programma chashi dopo 2 minuti o 2 settimane!
- Come identifico un bug dietro un altro bug?
- Non so quanto del software è stato eseguito e testato (code coverage)
- Praticamente nessun fuzzer oggi in circolazione utilizza i risultati degli input passati per costruirne poi di migliori

Tecniche avanzate

- Si utilizzano tecniche avanzate di code coverage (aka fuzzer tracking) per testare tutti i possibili percorsi che gli input possono raggiungere
- Tecniche di *automated protocol dissection* che utilizzano algoritmi genetici sono state create per fuzzare al meglio anche protocolli proprietari sconosciuti
- Algoritmi genetici sono anche usati per indirizzare la scelta degli input migliori in una data situazione in base a quelli passati: code coverage massimizzato
- Tecniche di *dynamic binary instrumentation* permettono di individuare la sorgente di un errore e non solo il suo verificarsi

Tecniche avanzate: white-box fuzzing

- No, non è un ossimoro, è un termine usato soprattutto da Microsoft che ha lavorato sul concetto nell'ultimo anno
- Non c'entra molto con l'Hybrid Analysis di Rafal Los
- Invio di input malformati con la garanzia però di coprire praticamente tutti i "percorsi" possibili grazie a tecniche di *dynamic test generation* e *symbolic execution*
- Molto più complesso e lento dell'approccio black-box ma molto più efficace (SAGE ha scoperto la vulnerabilità ANI)

Fuzzing & SDLC

- Microsoft sin dal 2004 ha integrato avanzate tecniche di fuzzing nel proprio Security Development Lifecycle (si è accorta che correggere una vulnerabilità in un prodotto già pubblico le costava milioni di $$$, mica per altro :P)
- Chiedere maggiore collaborazione ai security specialists durante il SDLC potrebbe essere utile (la segnalazione di una vulnerabilità in Firefox 3 poche ore dopo il rilascio evidenzia però il problema del security business)
- Il costo per patchare un software cresce più che linearmente all'avanzare del suo sviluppo

Conclusioni

- Le tecniche di fuzzing sono sicuramente il metodo più efficace per la ricerca di vulnerabilità (molte, non tutte!)
- Dopo che una vulnerabilità vi siete divertiti a trovarla, potete godere ancora di più exploittandola, se possibile!
- Esistono alcune soluzioni commerciali per il fuzzing: beSTORM, Codenomicon, Mu Dynamics, Holodeck (personalmente mi son trovato bene con beSTORM)

Links utili

Trovate queste slide all'indirizzo:

http://trivero.secdiscover.com/hat02.pdf

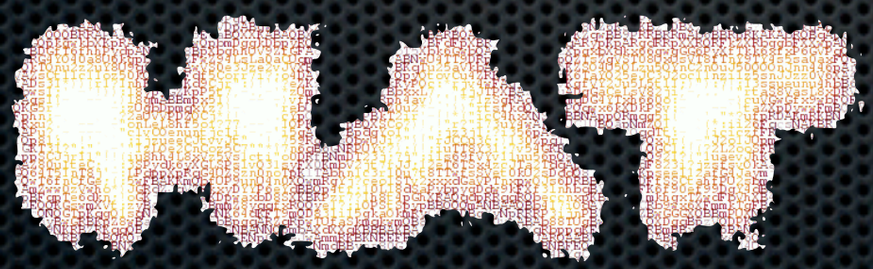
- h71028.www7.hp.com/ERC/cache/571092-0-0-0-121.html
- www.fuzzing.org

Fuzzing Pack for HAT's Attenders (esclusi quelli nella stanza Gogol :P)

+ 24 PAPERS

+ 81 FUZZERS =

Domande?



Relatore: Alberto Trivero

e-mail: a.trivero@secdiscover.com