TECHNISCHE UNIVERSITÄT DRESDEN

Professur Datenschutz und Datensicherheit

Department of Computer Science, Institute for Systems Architecture, Chair of Privacy and Data Security

Pentestlab — Brute-Force-Angriffe

dud.inf.tu-dresden.de

Stefan Köpsell (stefan.koepsell@tu-dresden.de)

Brute-Force — Überblick

brute force – rohe Gewalt / Brachialgewalt

generelle Idee:

- Ausprobieren vieler / aller möglichen Lösungskandidaten

Vorteil:

- in der Regel einfach umzusetzen

Nachteil:

- für viele Probleme sehr aufwendig (Laufzeit / Ressourcenverbrauch)

Brute-Force — ein Beispiel

Problem: Faktorisierung von $n \in \mathbb{N}$

Brute-Force-Algorithmen:

a) für alle $i \in \mathbb{N}, 1 < i < n$
   teste: $i \mid n$?

b) für alle $i \in \mathbb{N}, 1 < i < \frac{n}{2}$
   teste: $i \mid n$?

c) für alle $i \in \mathbb{N}, 1 < i < \sqrt{n}$
   teste: $i \mid n$?

d) für alle $i \in \mathbb{P}, \ 1 < i < \sqrt{n}$
   teste: $i \mid n$?

brute-force – Durchprobieren ***aller*** möglichen Lösungskandidaten

➔ was zählt zu „*alle*“?

➔ Effizienz / Komplexität eines Brute-Force-Algorithmus ein mögliches Gütekriterium

$$x^2 + 4x = n \ ?$$

Brute-Force—allgemeine Überlegungen

- Problemeigenschaften:
  - es muß etwas geben, was man Durchprobieren kann
  - Abbruchkriterium (für Erfolgsfall) notwendig:
    - Lösung ist überprüfbar
      - Beispiel: Faktorisierung
    - IT-System: Systemverhalten
      - Beispiel: Programmabsturz, erfolgreiche Anmeldung
    - zeitliche Begrenzung

Brute-Force-Aufwand

…Folien von Security & Cryptography I (Prof. Strufe)

How large is large? (Some context)

Reference Numbers Comparing Relative Magnitudes			
Reference	***Magnitude***		
Seconds in a year		$\approx 3 * 10^7$	
Seconds since creation of solar system		$\approx 2 * 10^{17}$	$\approx 4.6 * 10^9$ y
Clock cycles per year (50 MHz computer)		$\approx 1.6 * 10^{15}$	
Instructions per year (i7 @ 3.0 GHz)	$\approx 2^{63}$	$\approx 7.15 * 10^{18}$	
Binary strings of length 64	2^{64}	$\approx 1.8 * 10^{19}$	
Binary strings of length 128	2^{128}	$\approx 3.4 * 10^{38}$	
Binary strings of length 256	2^{256}	$\approx 1.2 * 10^{77}$	
Number of 75-digit prime numbers		$\approx 5.2 * 10^{72}$	
Number of 80-digit prime numbers		$\approx 5.4 * 10^{77}$	
Electrons in the universe		$\approx 8.37 * 10^{77}$	

Attacking Cryptography: Brute Force Attack

Brute force attack, try all keys until intelligible plaintext found:

- Every crypto algorithm (save: OTP) can be attacked by brute force
- On average, half of all possible keys will have to be tried

Average Time Required for Exhaustive Key Search

Key Size [bit]	*Number of keys*	*Time required at 1 encryption / µs*	*Time required at 10^6 encryption / µs*
32	$2^{32} = 4.3 * 10^9$	2^{31} µs = 35.8 minutes	2.15 milliseconds
56	$2^{56} = 7.2 * 10^{16}$	2^{55} µs = 1142 years	10.01 hours
128	$2^{128} = 3.4 * 10^{38}$	2^{127} µs = $5.4 * 10^{24}$ years	$5.4 * 10^{18}$ years
(Vigenère 26 chars	$26! = 4 * 10^{26}$	2^{88} µs = $6.4*10^{12}$ years	$6.4*10^6$ years)

i7 could get to around 10^4 encryptions/µs
GPU can perform around $7x10^5$ hashes

Time since human/chimpanzee lines diverged:
$5×10^6$ years,
Homo sapiens: $5×10^4$ years

Brute-Force— allgemeine Überlegungen

- Strategien bezüglich Durchprobieren:
  - systematisch / vollständig / deterministisch:
    - alle Elemente einer endlichen Menge an Hand einer Wohlordnung
      - "natürliche" / "kanonische" Ordnung
      - gewichtet beispielsweise an Hand von Auftrittswahrscheinlichkeiten
    - *exhaustive search*
    - Beispiele:
      - Schlüsselsuche im Bereich Kryptographie
      - Paßwortsuche
  - indeterministisch:
    - Zufall beeinflußt Kandidatenauswahl
    - ggf. nicht vollständig
    - *fuzzing*

Randomisierte Algorithmen

- randomisierter / stochastischer /probabilistischer Algorithmus:
  - Zufall beeinflußt Abarbeitung des Algorithmus
    - Parameterwahl
  - **mehrfache Ausführung kann zu unterschiedlichen Ergebnissen führen**

- Las-Vegas-Algorithmus:
  - liefert nie ein falsches Ergebnis
  - Unterarten:
    - a) liefert immer richtiges Ergebnis, Laufzeit im worst-case sehr groß
    - b) Algorithmus liefert kein / Teilmenge der Ergebnisse, Laufzeit beschränkt
  - Beispiel: Faktorisierung mit zufälliger Faktorenauswahl

- Monte-Carlo-Algorithmus:
  - kann falsches Ergebnis liefern
    - Schranke für Fehlerwahrscheinlichkeit bestimmbar

Fuzzing

- ursprüngliche (?) Idee:
  - 1988 als Teil einer Praktikumsaufgabe am Computer Science Department der Universität von Wisconsin-Madison; Betreuer: Bart Miller:

    "The goal of this project is to evaluate the robustness of various UNIX utility programs, given an unpredictable input stream. This project has two parts. First, you will build a fuzz generator. This is a program that will output a random character stream. Second, you will take the fuzz generator and use it to attack as many UNIX utilities as possible, with the goal of trying to break them."
    [http://pages.cs.wisc.edu/~bart/fuzz/CS736-Projects-f1988.pdf]

  - Konzept [http://pages.cs.wisc.edu/~bart/fuzz/]:
    - The input is random. We do not use any model of program behavior, application type, or system description.
    - Our reliability criteria is simple: if the application crashes or hangs, it is considerd to fail the test, otherwise it passes.

Fuzzing

- häufig angewendet im Rahmen von Black-Box-Tests / Untersuchungen

- "intelligentere" Anwendung in White-Box / Gray-Box Situationen möglich
  - modellgesteuerte Generierung der zufälligen Eingabedaten
    - abgeleitet von aufgezeichnete Daten (Mutation)
    - aus Protokoll / Programm / Schnittstellenspezifikation
  - Beispiel: zufällige Eingabedaten mit (korrekt berechneter) Prüfsumme

- verwandte Begriffe: Lasttest / Streßtest
  - fault injection → insbesondere im Hardwarebereich schon früh eingesetzt

- Vielzahl an Werkzeugen existiert
  - **`fuzz`**, **`afl`**

- Vielfältige Einsatzmöglichkeiten
  - Robustheits- / "Korrektheits"-Test von Programmen
  - Test von Protokollimplementierung im Netzbereich
  - Hardwaretests
  - Sicherheitstest

Fuzzing

- Ziele:
  - unmittelbarer Erfolg
    - System zeigt aus Angreifersicht gewünschtes Verhalten
    - Beispiel:
      - Umgehung von Zugriffskontrollmechanismen durch Fuzzing von Eingabeparametern
  - mittelbarer Erfolg
    - System zeigt Verhalten, daß auf weitere Angriffsmöglichkeiten hindeutet
    - Beispiel: Systemabsturz → ggf. erfolgreiche Bufferoverflow / Stackoverflow-Angriffe möglich

Rowhammer

- Mark Seaborn, Thomas Dulien: „Exploiting the DRAM rowhammer bug to gain kernel privileges“, März 2015
  - basierend auf: Kim et al.: „Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors“

    ➔ Bit-Flip in Speicherzelle *X* durch wiederholten Zugriff auf Speicherzelle *Y*

```
code1a:
  mov [a], %eax  // copy data to address a
  mov [b], %ebx  // copy data to address b
  clflush (a)    // flush cache for address a
  clflush (b)    // flush cache for address b
  jmp code1a
```

  - funktioniert auch mit JavaScript…
  - und AMD-Przessoren: „ZENHAMMER: Rowhammer Attacks on AMD Zen-based Platforms“ (2024)

- Angriffsszenarien:
  - Manipulation von “sicherem Code” bei Sandboxing-Verfahren → Ausbruch aus Sandbox
    - demonstriert für Google Chrome Native Client
  - Betriebssysteme: Manipulation der Page Table
    - Read/Write Zugriffs-Beschränkungen
    - Abbildung virtuelle Adresse → physische Adresse

Native Client Exploit

[from: https://www.blackhat.com/docs/us-15/materials/us-15-Seaborn-Exploiting-The-DRAM-Rowhammer-Bug-To-Gain-Kernel-Privileges.pdf]

Safe instruction sequence:

```
andl $~31, %eax          // Truncate address to 32 bits
                         // and mask to be 32-byte-aligned.
addq %r15, %rax          // Add %r15, the sandbox base address.
jmp *%rax                // Indirect jump.
```

NaCl sandbox model:

- Prevent jumping into the middle of an x86 instruction
- Indirect jumps can only target 32-byte-aligned addresses

Exploit idea: `%eax` → `%ecx`

- Allows jumping to a non-32-byte-aligned address

Example

mem-addr: 48 b8 0f 05 eb 0c f4 f4 f4 f4

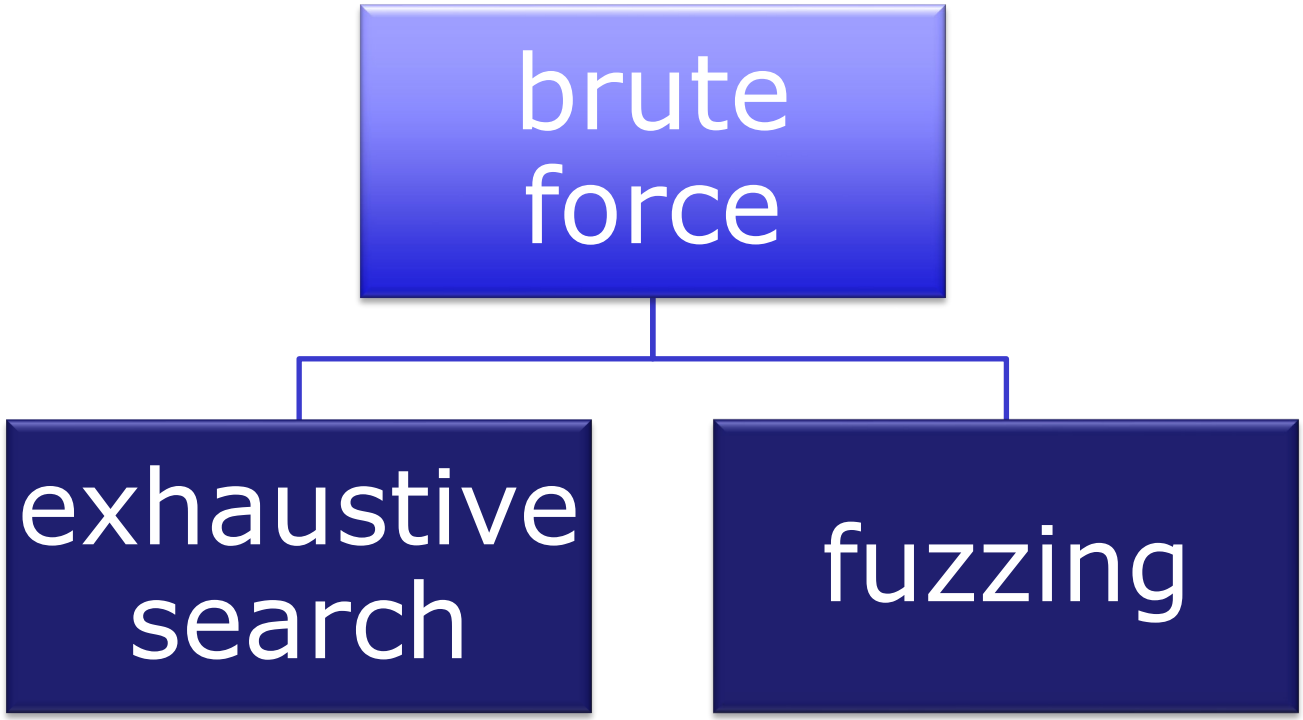
mem-addr: `movabs $0xf4f4f4f40ceb050f, %rax`

```
mem-addr+2: 0f 05    syscall
mem-addr+4: eb 0c    jmp
mem-addr+6: f4       hlt
```

//shouldn't this be 0f 34?

Brute-Force — Begrifflichkeiten

- zufriedenstellende Definitionen / umfassende Terminologie / Systematik ist mir nicht bekannt
- Diskussionsvorschlag



Fuzzing—Spielarten

Werkzeug	primäres Ziel	Beispiel
Hardware	Hardware	Hardwaretest mittels fault injection (Strahlung, Spannung), Angriffe auf Smartcards
Hardware	Software	ASIC-basierte Fuzzing-Angriffe (?)
Software	Hardware	Rowhammer
Software	Software	„klassisches Fuzzing“, Software-basierte Robustheitstest

Brute-Force-Angriffe

Brute-force Begrifflichkeiten — ein zweiter Ansatz

Brute-Force-Angriffe		Eingabe	
		deterministisch	zufällig
Ausgabe	deterministisch	exhaustive search	
	zufällig		

Brute-force Begrifflichkeiten — ein zweiter Ansatz

Brute-Force-Angriffe		Eingabe	
		deterministisch	zufällig
Ausgabe	deterministisch	exhaustive search	fuzzing
	zufällig	Beachten: • (zeitlicher) Aufwand • Aussagekraft des Testergebnisses • Testwiederholungen notwendig	

Brute-Force— weitere Aspekte

- Adaptivität
  - nachfolgende Eingabe bestimmt durch beobachtete Ausgabe
  - Bezugspunkte zum reverse engineering
    - Beispiel: Aufbau Nachrichtenformat

- Meßbarkeit des Erfolgs
  - bisher: Erfolg muß meßbar / beobachtbar sein
    - sinnvoll / hilfreich / notwendig für Penetration-Test
  - Brute-Force-Angriff:
    - Erfolg muß nicht meßbar sein
      - Beispiel

```
frontend()
  {
    int 32 r=rand();
    loop()
      {
        i=inputFromNet();
        if(i==r)
          explode(backend);
      }
  }
```

Literatur

- Evron et al.: „Open Source Fuzzing Tools“, Elsevier, 2007
- Takanen, Demott, Miller: „Fuzzing for Software Security Testing and Quality Assurance“, Artech House, 2008

Penetrationstest Anwendungsfall

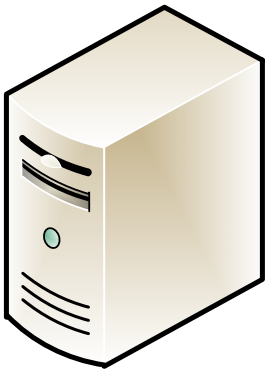
Brute-Force im Bereich Paßwörter

Password based authentication

- Simple approach

User

(dog,bone)



Server

yes

grant access

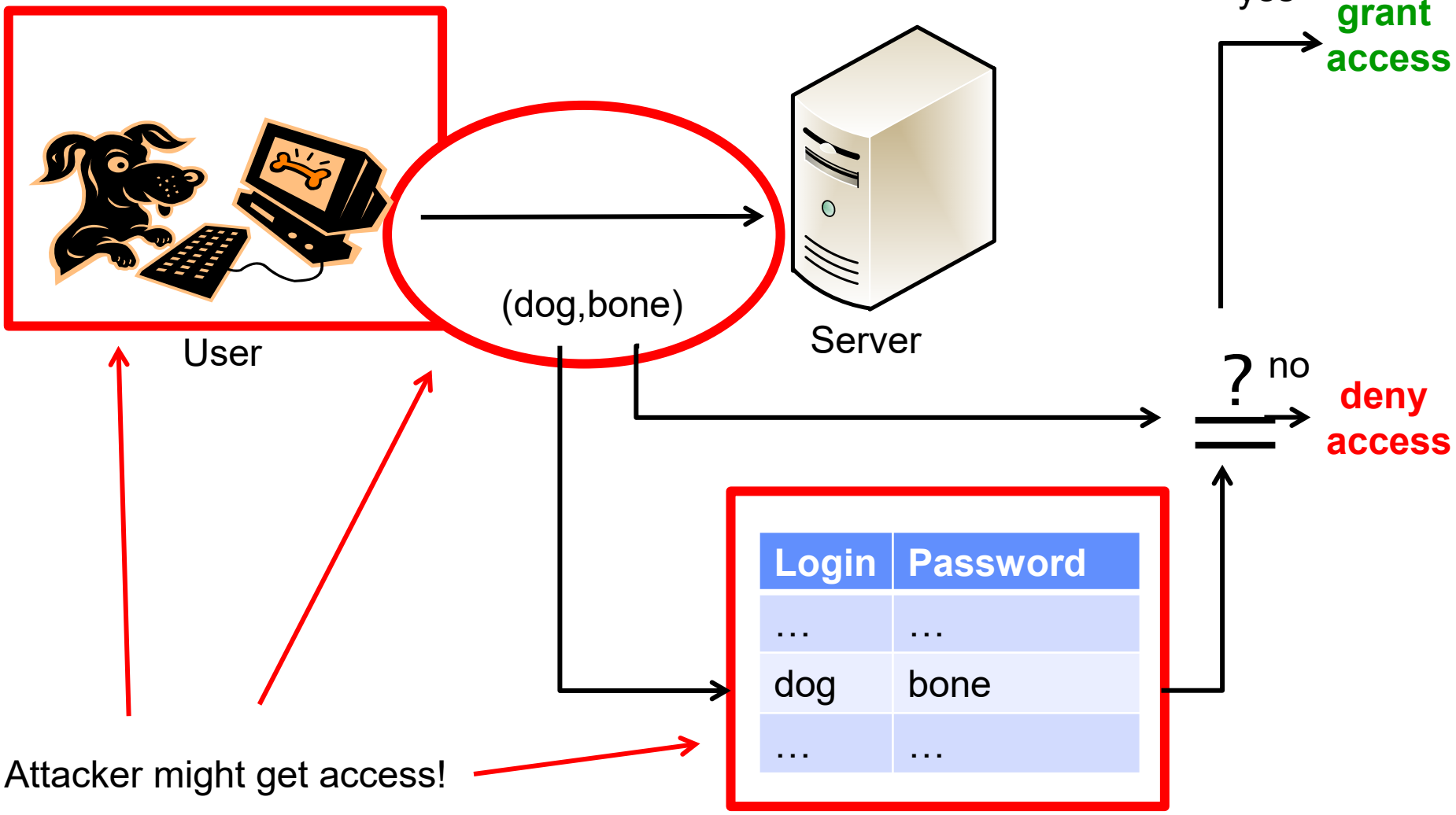
? no

deny access

Login	Password
…	…
dog	bone
…	…

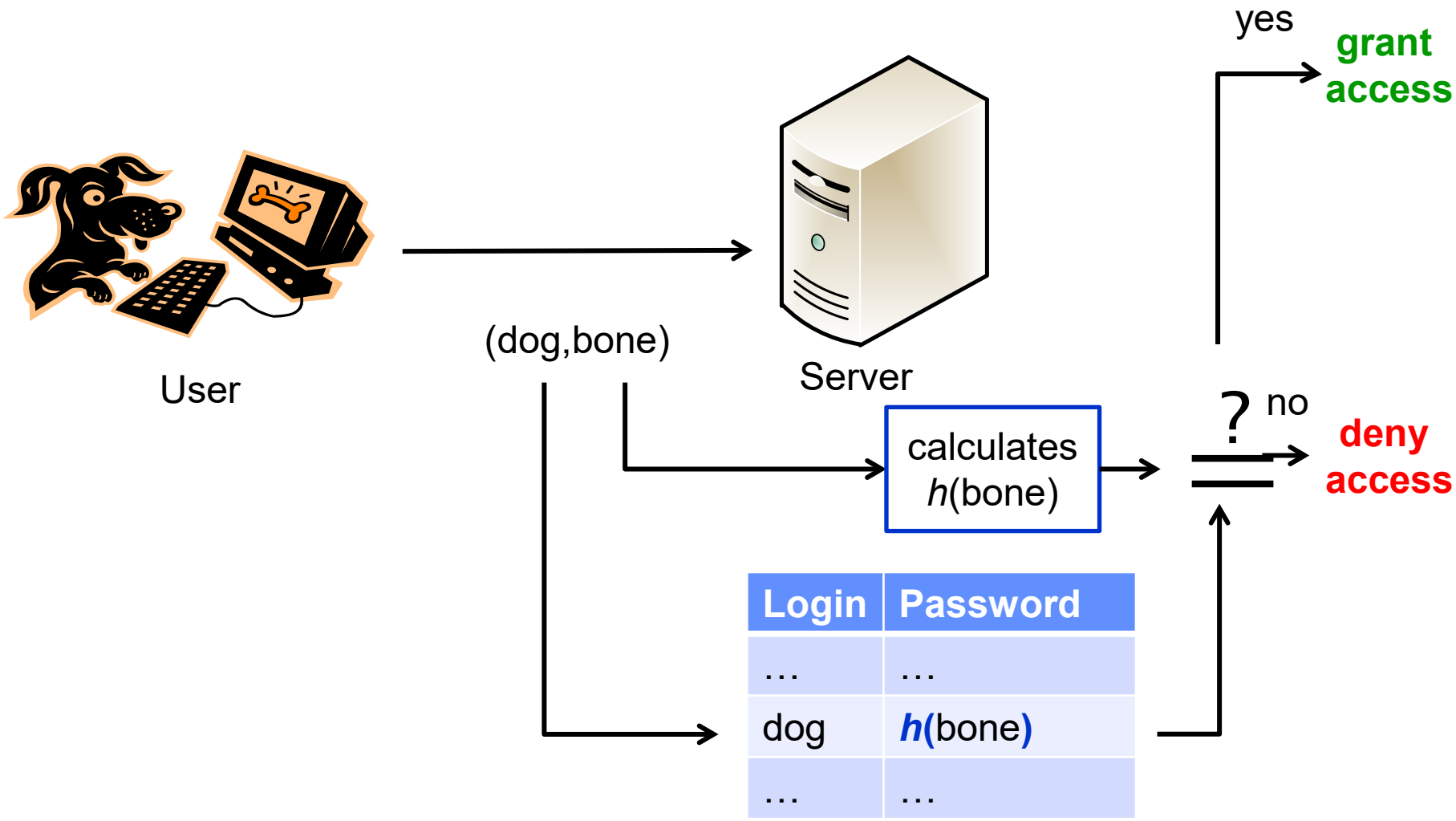
Password based authentication

- Simple approach – **security problems**



Password based authentication

- Enhanced approach using one way (hash) functions



One-way functions – cryptographic hash functions

- One-way function *f:*
  - calculating $f(x)=y$ is easy
  - calculating $f^{-1}(y)=x$ is hard
    - computation / storage
  - open question: Do one-way functions exist?
- Cryptographic hash function *h*
  - might have different properties depending on the use case
  - collision resistance:
    - it is hard to find *x*, *y* with $h(y)=h(x)$ and $y \neq x$
    - note: *h* is usually not *collision free*, because $|h(x)| \ll |x|$
  - preimage resistance / one-way function / secrecy
    - given $h(x)$ it is hard to find *x*
  - second-preimage resistance / weak collision resistance / binding
    - given x, $h(x)$ it is hard to find y with $h(y)=h(x)$ and $y \neq x$
  - Note:
    - *h* is not necessarily a “random extractor”
    - only one of “secrecy” and “binding” can be information theoretic secure

Examples for cryptographic hash functions

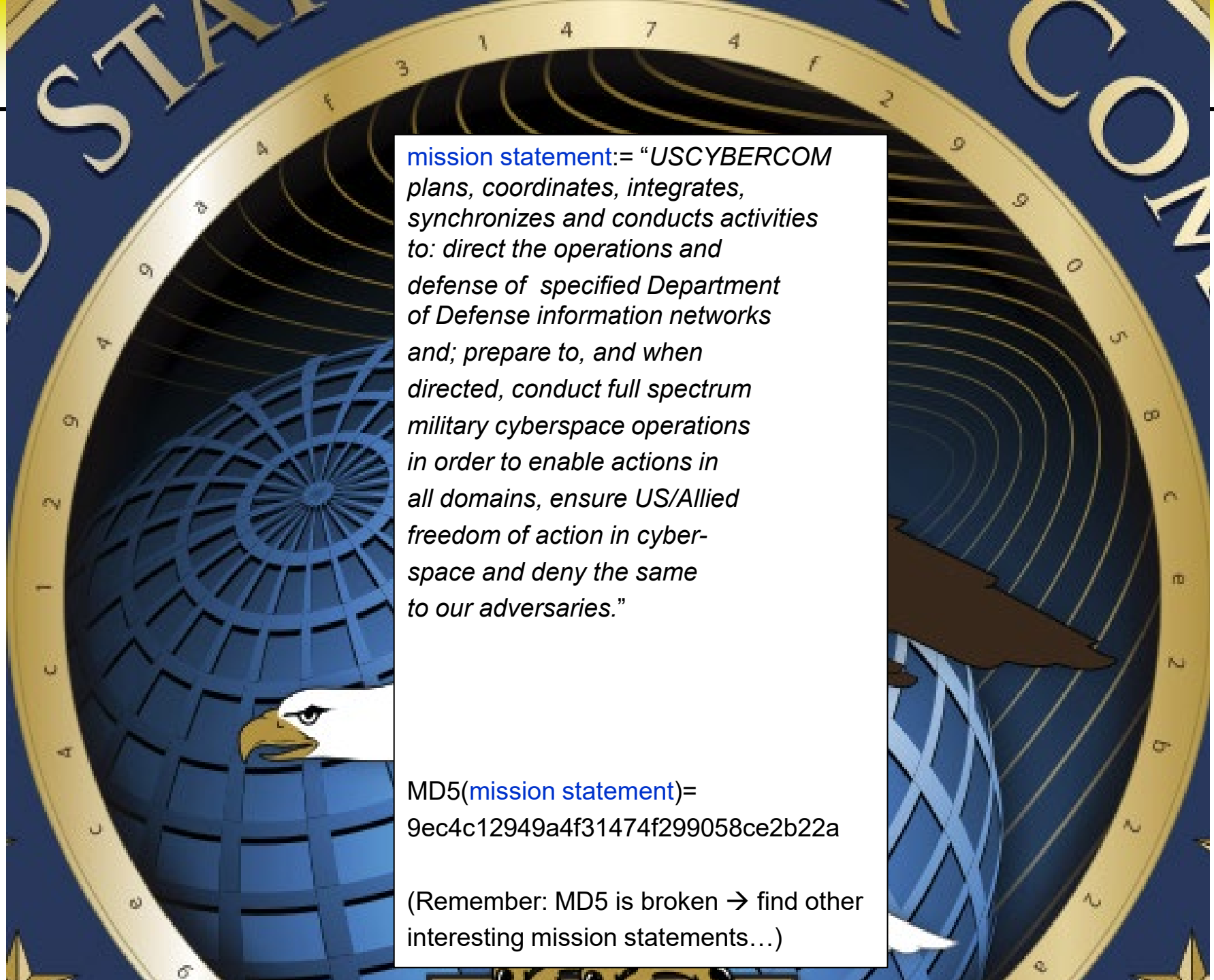
- MD5
  - Message-Digest Algorithm
  - developed by Ronald Rivest (April 1992)
  - produces 128 bit hash values
  - can process arbitrary long inputs
  - **today MD5 is broken!**
- SHA-1
  - Secure Hash Standard
  - published 1993 as FIPS PUB 180 by US NIST
  - produces 160 bit hash values
  - **today SHA-1 is insecure!**
- SHA-2
  - set of hash functions, with hash values of 224, 256, 384, 512 bit
  - published 2001 as FIPS PUB 180-2 by NIST (current version: FIPS 180-4)
  - **SHA-2 hash functions are believed to be secure**
- SHA-3
  - result of the NIST Cryptographic Hash Algorithm Competition started November 2007
  - 3 selection rounds, 5 finalists
  - October 2012: Keccak is winner
  - FIPS 202: “SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions” (08/15)

MD5 Hash in the Wild

- United States Cyber Command (USCYBERCOM)
  - mission statement:= "*USCYBERCOM plans, coordinates, integrates, synchronizes and conducts activities to: direct the operations and defense of specified Department of Defense information networks and; prepare to, and when directed, conduct full spectrum military cyberspace operations in order to enable actions in all domains, ensure US/Allied freedom of action in cyberspace and deny the same to our adversaries.*"

MD5 Hash in the Wild

mission statement:= "*USCYBERCOM plans, coordinates, integrates, synchronizes and conducts activities to: direct the operations and defense of specified Department of Defense information networks and; prepare to, and when directed, conduct full spectrum military cyberspace operations in order to enable actions in all domains, ensure US/Allied freedom of action in cyber-space and deny the same to our adversaries.*"



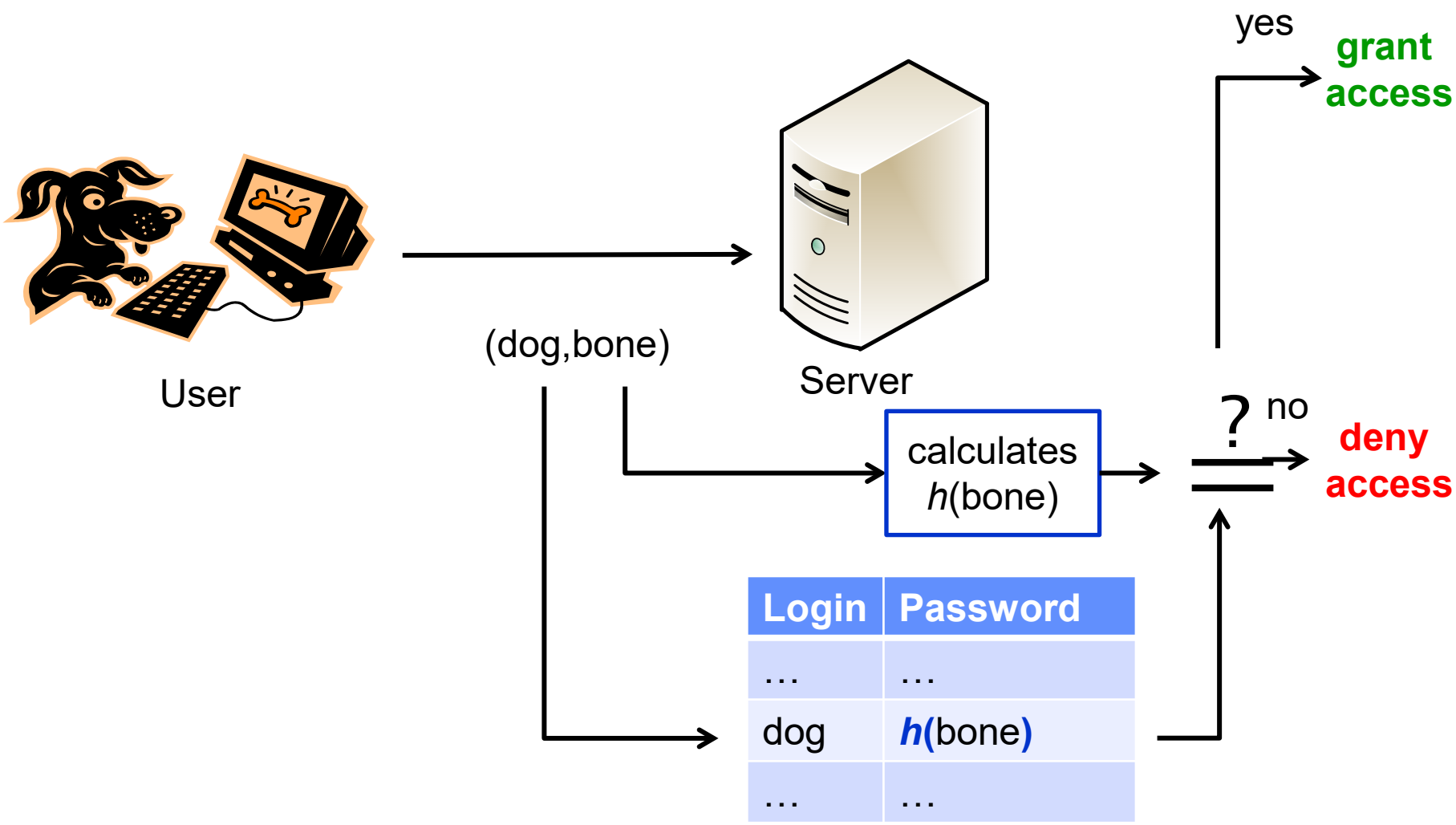
mission statement:= "*USCYBERCOM plans, coordinates, integrates, synchronizes and conducts activities to: direct the operations and defense of specified Department of Defense information networks and; prepare to, and when directed, conduct full spectrum military cyberspace operations in order to enable actions in all domains, ensure US/Allied freedom of action in cyber-space and deny the same to our adversaries.*"

MD5(mission statement)=
9ec4c12949a4f31474f299058ce2b22a

(Remember: MD5 is broken → find other interesting mission statements…)

Password based authentication

- Enhanced approach using one way (hash) functions

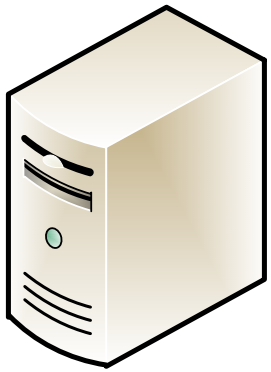


Password based authentication

- Enhanced approach using one way (hash) functions

User

(dog,bone)



Server

yes

grant access

calculates *h*(bone)

? no

deny access

Login	Password
…	…
dog	*h*(bone)
…	…

Slightly reduced risk, if attacker gets access.

Remaining problems of password based authentication based one way functions

- Brute Force attack
  - function *h*() is public
  - value of *h*(*x*) is known to the attacker

  ➔ try all possible values for *x*

  Considerations:
    - usually >> 1 Mio. *h*(x)/s on ordinary hardware
    - assumption: password uses only small letters
    - password length = 8

Login	Password
…	…
dog	*h*(bone)
…	…

time needed: $\frac{26^8}{1\,000\,000 \cdot 60 \cdot 60} \approx 58\text{h}$

- first countermeasures:
  - limit false attempts
- first password rules:
  - use a large alphabet (small and capitalised letters, numbers, specials)
  - use a long password

Remaining problems of password based authentication based one way functions

- first password rules:
  - use a large alphabet
    - (small, capitalised letters, numbers, specials)
    - time needed: $\frac{(26+26+10+30)^8}{1\,000\,000 \cdot 60 \cdot 60 \cdot 24 \cdot 365.25} \approx 162a$
  - use a long password

Login	Password
…	…
dog	*h*(bone)
…	…

- remaining possible attacks:
  - increase in computation power
    - distributed approach
    - GPU
    - Moore's law
  - pre-computation:
    - attacker creates lockup table
    - search time (example above):

      $\mathrm{ld}\left((26+26+10+30)^8\right) < 53$ comparisons

Remaining problems of password based authentication based one way functions

- remaining possible attack:
  - pre-computation:
    - attacker creates lockup table
    - search time (example above):

      $\mathrm{ld}\left((26+26+10+30)^8\right)<53$ comparisons
    - problem: storage space
    - Example:
      - size of MD5 hash: 128 bit
      - number of hashes: $(26+26+10+30)^8$=5 132 188 731 375 616
      - storage
        - space: 5 132 188 731 375 616 · 128 bit ≈ 75 000 TByte
        - price: ≈ 2 Mio €
    - solution: storage space / computation tradeoff
      - Martin E. Hellman: “A Cryptanalytic Time – Memory Trade-Off”

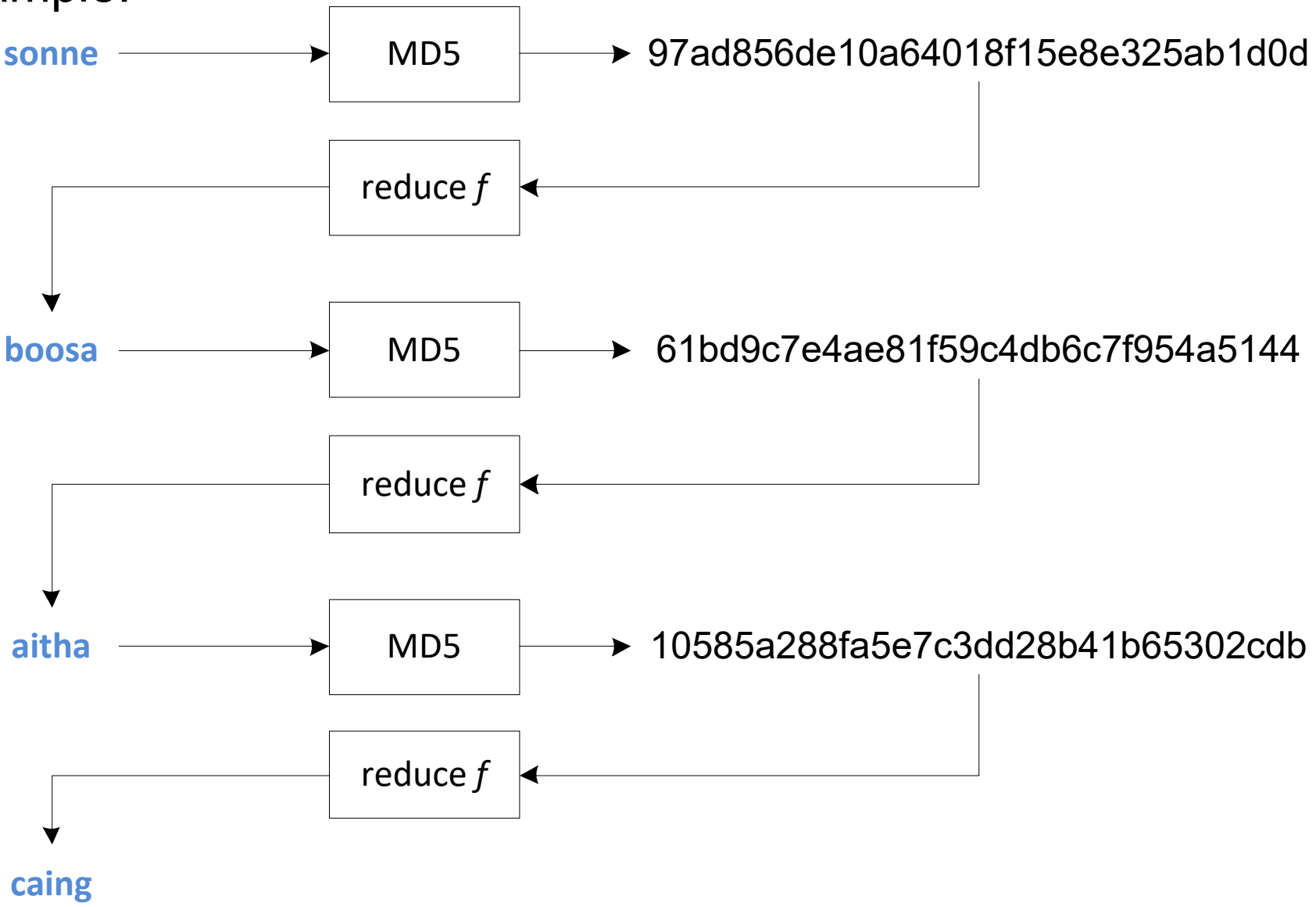
Login	Password
…	…
dog	*h*(bone)
…	…

Cryptanalytic Time – Memory Trade-Off

- main idea:
  - store only certain parts of the lookup table
  - regenerate the missing parts on demand
- requires “reduce” function *f*
  - *f*: H → P (H: set of hash values, P: set of passwords)
  - note: *f* is NOT the inverse of *h*
- general procedure:
  - calculate a chain of hash and reduce function calls
    - p → h() → f() → h() → f() → h () … → f() → p’
  - store first and last value in a table
    - sort by the last value
  - length of chain influences Time – Memory trade-off

Cryptanalytic Time – Memory Trade-Off

- Example:



Cryptanalytic Time – Memory Trade-Off

- 2nd example
  - breaking of 4 digit PINs
  - *h*(x):= (x·7807) mod 16157
  - *f*(x) := x mod 9000 + 1000

- PIN-table:

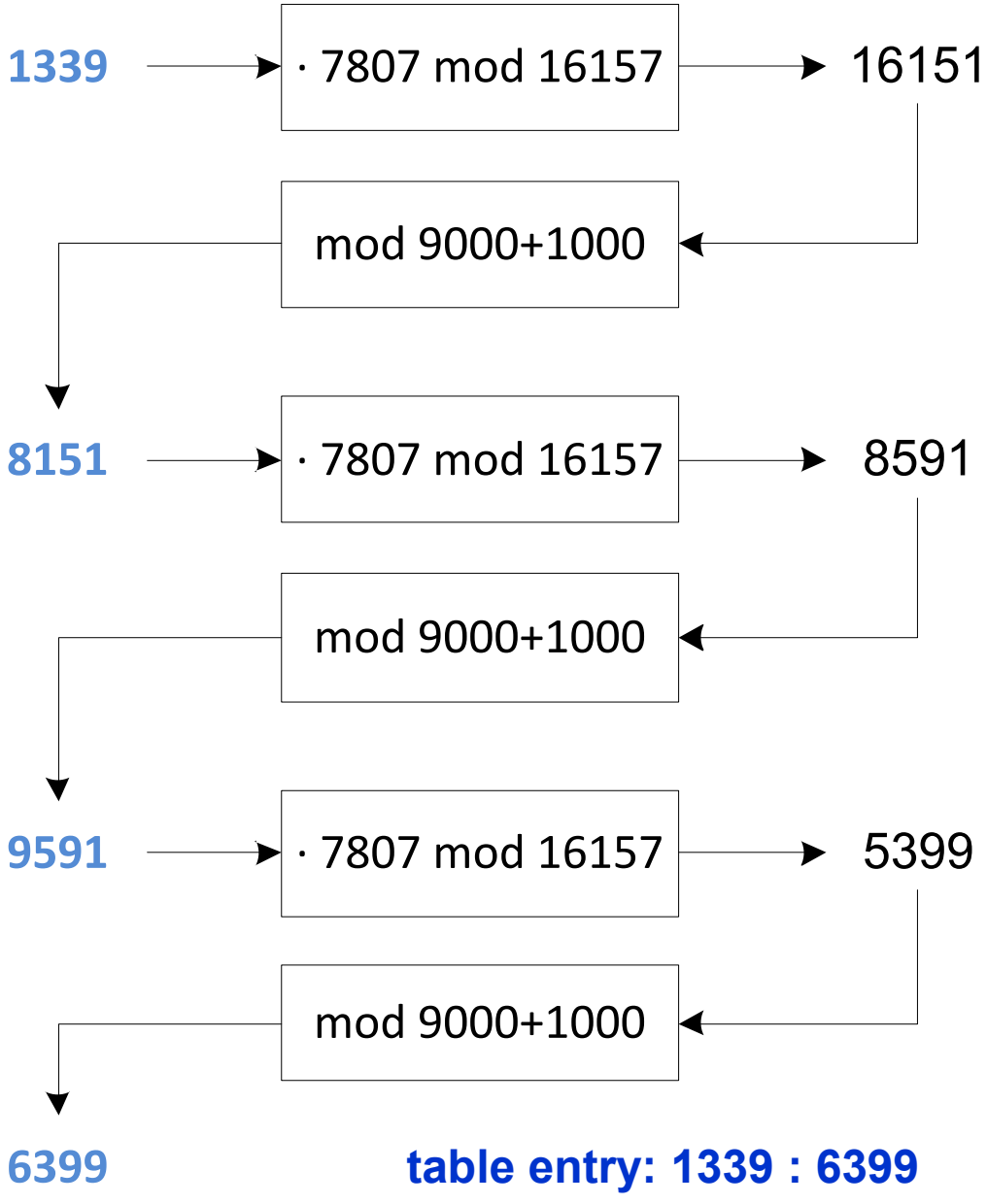
1309–9139–7018–**2139**

2439–9327–4447–**4493**

1084–4677–6676–**5207**

1339–8151–9591–**6399**

3128–8069–6697–**7584**

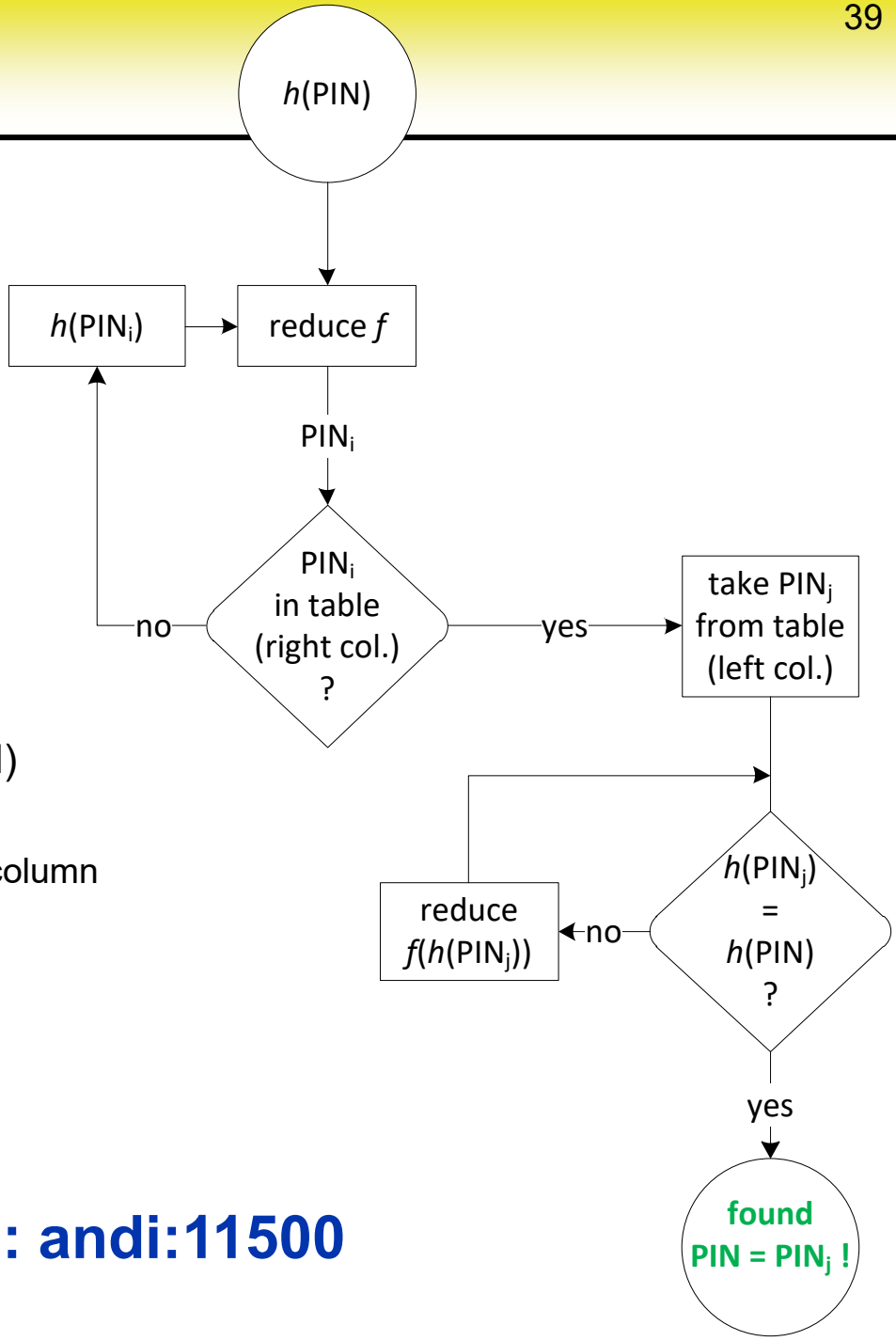


Cryptanalytic Time – Memory Trade-Off

- PIN-table:

2439-9327-4447-4493-**1003**
1084-4677-6676-5207-**1037**
3128-8069-6697-7584-**1040**
2824-9820-7932-3500-**4013**
1339-8151-9591-6399-**7706**
1309-9139-7018-2139-**9992**

- Breaking a PIN:
  - Goal: find PIN for given hash value *h*(PIN)
  - Algorithm:
    - 1. hash / reduce until value is in the right column
    - 2. take left column value
    - 2. hash / reduce until PIN is found
  - *h*(x):= (x·7807) mod 16157
  - *f*(x) := x mod 9000 + 1000



Try to break Andi‘s password: andi:11500

Remaining problems of password based authentication based one way functions

- remaining possible attack:
  - pre-computation:
    - attacker creates lockup table
    - search time (example above):

      $\mathrm{ld}\left((26+26+10+30)^8\right)<53$ comparisons
    - problem: storage space
    - Example:
      - size of MD5 hash: 128 bit
      - number of hashes: $(26+26+10+30)^8$=5 132 188 731 375 616
      - storage
        - space: 5 132 188 731 375 616 · 128 bit ≈ 75 000 TByte
        - price: ≈ 2 Mio €
    - solution: storage space / computation tradeoff
      - Martin E. Hellman: “A Cryptanalytic Time – Memory Trade-Off”
    - today: use the power of the Internet / Cloud
      - http://www.freerainbowtables.com/

Login	Password
…	…
dog	*h*(bone)
…	…

Remaining problems of password based authentication based one way functions

- remaining possible attack:
  - pre-computation
- countermeasure:
  - salt & pepper!
  - $h(x) \rightarrow h(\mathbf{salt},x) \rightarrow h(\mathbf{salt},f(x,\mathbf{pepper}))$
  - salt:
    - long (e.g. 128 bit) random value
    - some part could be stored together with password (i.e. 104 bit)
    - some part could not be stored at all (i.e. 24 bit)
      - verification: iterate over all possible salt values
  - pepper:
    - random value
    - stored separate from password list
    - unique per system or per password
  - additional: repeated hashing

➔ pre-computation has to be done *for each possible salt & pepper*

Login	Password
…	…
dog	*h*(bone)
…	…

Remaining problems of password based authentication based one way functions

- remaining possible attack:
  - **dictionary attack**
  - problem: people do not chose passwords **randomly**
  - often names, words or predictable numbers are used
  - http://www.whatsmypass.com/the-top-500-worst-passwords-of-all-time
  - attacker uses dictionaries for brute force attack
  - prominent program: *John the Ripper*
    - supports dictionary attacks and password patterns

Login	Password
…	…
dog	*h*(salt,bone)
…	…

- possible solutions:
  - enforce password rules
    - consider usability
  - pre-check passwords (e.g. using John)
  - train people to “generate” good passwords
    - Example: sentence → password
    - “This is the password I use for Google mail” → “TitpIu4Gm”

Hashfunktionen zum Speichern von Paßwörtern

übliche Anforderung an Hashfunktion:

- möglichst effiziente Verarbeitung großer Datenmengen

Problem:

- fördert Effizienz von Brute-Force-Angriffen

Spezialfall Paßwörter:

- Eingabe typsicherweise klein (<512 bit)
- gewisse Wartezeit für Login-Durchführung akzeptabel
  - ~ 1 Sekunde

  ➔ Paßwort-Hashfunktion muß nicht besonders effizient sein

Erschweren von Brute-Force-Angriffen:

- Paßwort-Hashfunktion darf nicht *effizient* implementierbar sein

Hashfunktionen zum Speichern von Paßwörtern

Paßwort-Hashfunktion darf nicht *effizient* implementierbar sein

- Software-Implementierungen:
  - Berücksichtigung aktueller CPU-Fähigkeiten
    - Multi-Core / Multi-Threaded
    - SIMD / Vector Extensions (AVX512)
    - Krypto-Erweiterungen (AES / SHA Befehle)
    - Cache-Größen (L1, L2, L3 Cache)
    - Branch Prediction
    - …
  - Berücksichtigung von verbreiteter „Spezial-Hardware“
    - Grafikkarten

- Hardware-Implementierungen
  - FPGA
  - spezial ASICs
    - Bitcoin-Mining

- zukunftsfähig
  - leichte Anpassung (Parametrisierung) an zukünftige (Hardware)-Verbesserungen

Hashfunktionen zum Speichern von Paßwörtern

Praktische Umsetzungen

- **bcrypt**
  - Niels Provos, David Mazières: „A Future-Adaptable Password Scheme“, USENIX, 1999
  - basiert auf Blowfish
    - symmetrische Blockchiffre

```
round_keys=EksBlowfishSetup(cost, salt, input) // ineffizient!
hash="OrpheanBeholderScryDoubt" // 3 x  64-bit blocks
loop (64)
    {
        hash=Blowfish_ECB(round_keys,hash)
    }
```

  - guter Schutz gegen Software / GPU basierte Brute-Force-Angriffe
  - schlechter Schutz gegen ASIC-basierte Brute-Force-Angriffe

Hashfunktionen zum Speichern von Paßwörtern

Praktische Umsetzungen

- **PBKDF2** (Password-Based Key Derivation Function 2)
  - ursprünglich in RSA Laboratories PKCS#5-Standard
    - gedacht für Ableitung symmetrischer Schlüssel aus Paßwort
    - übernommen als RFC 2898
    - anerkannt vom NIST in SP 800-132 (Dezember 2010)
  - *h*=PBKDF2 (passwd, salt, iterations)

```
{
  h=Hash(passwd||salt||iterations);
  loop(iterations-1)
    {
       h=h XOR Hash(passwd||h);
    }
  return h;
}
```

  - guter Schutz gegen CPU-basierte Software Brute-Force-Angriffe
  - schlechter Schutz gegen ASIC/FPGA/GPU-basierte Brute-Force-Angriffe

PBKDF2— Speicherverbrauch erhöhen

- *h*=PBKDF2 (passwd, salt, iterations)

```
{
  i=0
  h[i++]=Hash(passwd||salt||iterations);
  loop(iterations-1)
    {
       h[i+1]=h[i] XOR Hash(passwd||h[i]);
       i++;
    }
  sort(h[]);
  i=0;
  loop(iterations/2)
    {
      res=res + h[i] * h[i+1];
      i+=2;
    }
  return res;
}
```

Warning: Hand crafted crypto! - unverified -

Hashfunktionen zum Speichern von Paßwörtern

Praktische Umsetzungen

- **`scrypt`**
  - Colin Percival: “Stronger Key Derivation Via Sequential Memory-Hard Functions”, 2009
  - veröffentlicht in RFC 7914
  - Ziel: Hardware-Implementierung kostspielig machen
  - Lösungsansatz:
    - Speicherbedarf für effiziente Umsetzung stark erhöhen
  - Umsetzung:
    - Algorithmus benötigt großen Vektor pseudozufälliger Elemente, auf die in pseudozufälliger Reihenfolge zugegriffen wird
  - Kosten parameterisierbar

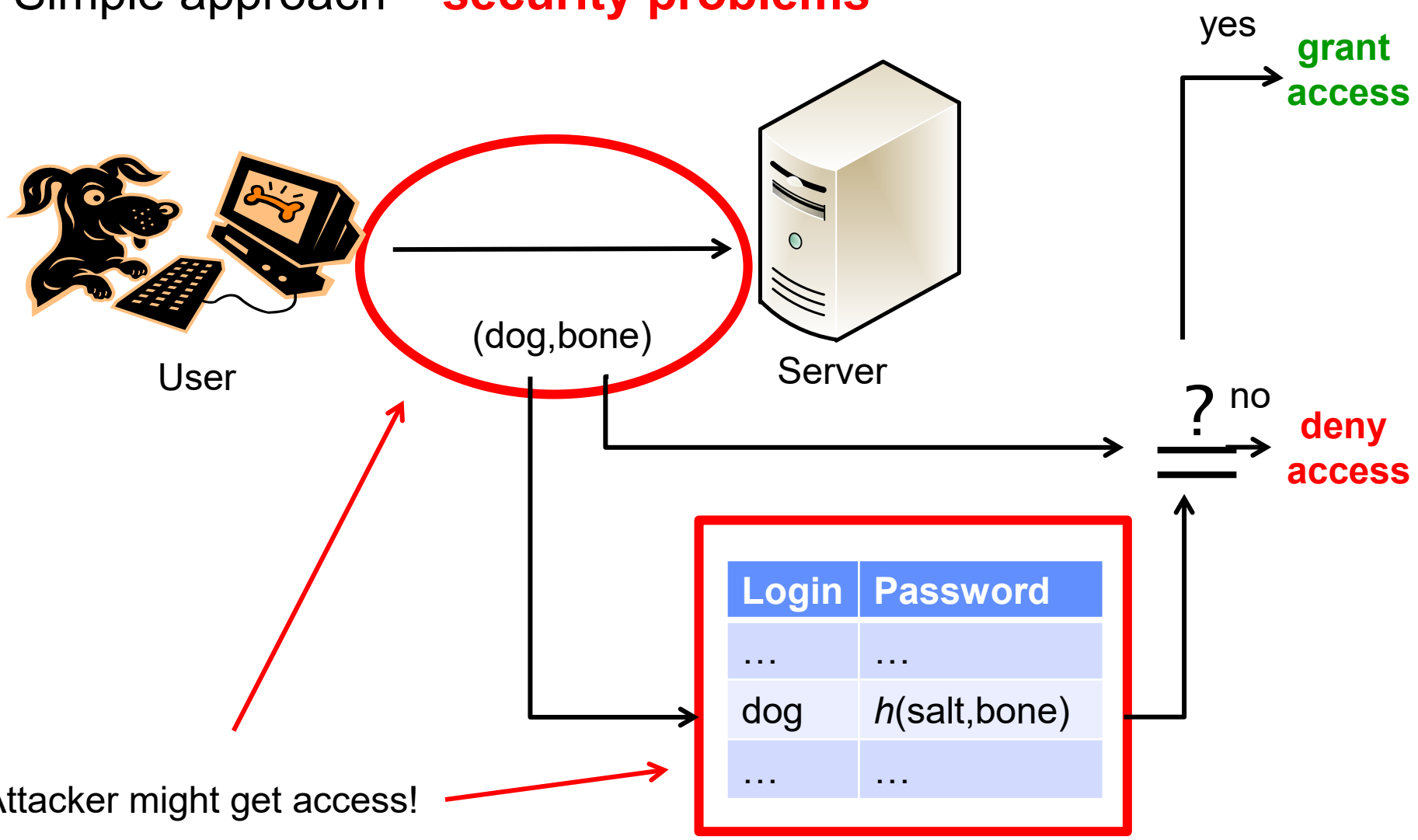
Hashfunktionen zum Speichern von Paßwörtern

Praktische Umsetzungen

- **Argon2**
  - Alex Biryukov, Daniel Dinu, Dmitry Khovratovich: “Argon2: the memory-hard function for password hashing and other applications”, 2015
  - Sieger der Password Hashing Competition (PHC)
    - Community getriebener Wettbewerb (2013-2015)
  - ähnliche Ziele / Überlegungen wie scrypt
  - bisher wenige bekannte Kryptanalyse

Password based authentication

- Simple approach – **security problems**



Password based authentication

- Simple approach – **security problems**

User

(dog,bone)

Server

yes

grant access

? no

deny access

Login	Password
…	…
dog	*h*(salt,bone)
…	…

Might be the attacker!

The Server as Attacker

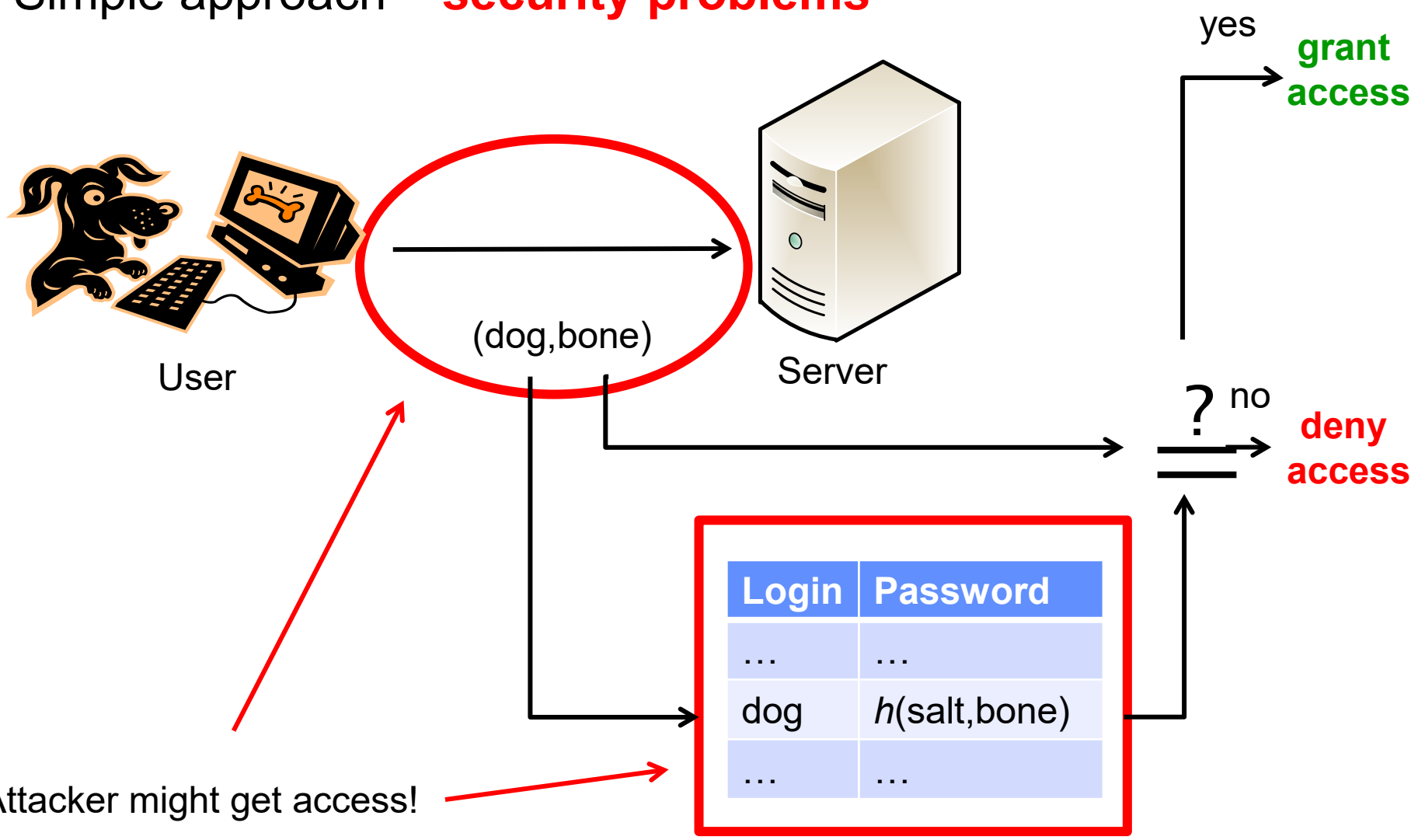
- … a new Web 2.0 service
- … for people which like city journeys
- … find cool cities and places like shops, restaurants, hotels etc.
- ... information from globe-trotters for globe-trotters
- … they can share their knowledge after secure login
- So that's wrong?

➔ It collects (username,password) and tries to login into other popular services like Gmail, Twitter, eBay, Amazon etc.

password rule: never "reuse" passwords!

Password based authentication

- Simple approach – **security problems**



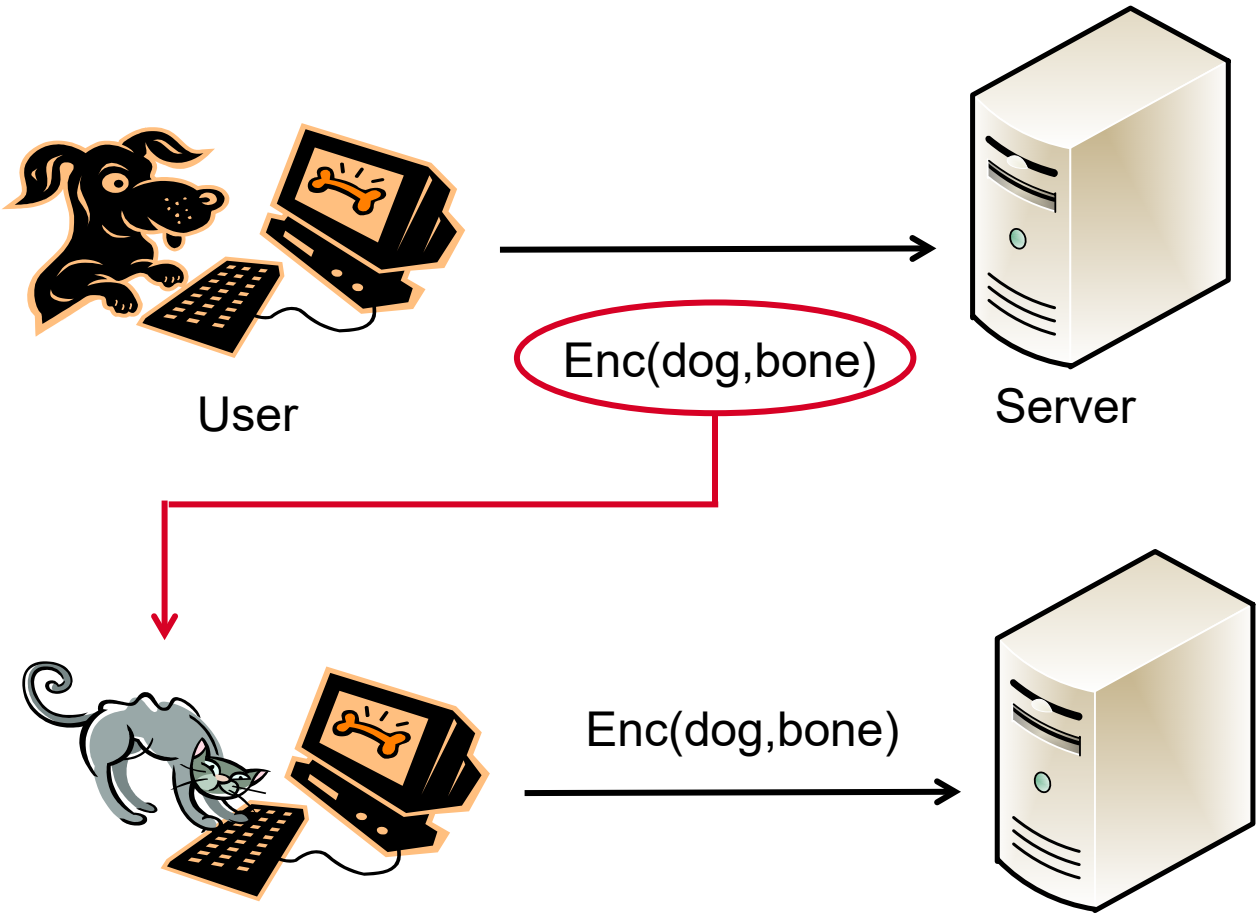
Password based authentication

- **security problems**

- possible solution:
  - encrypt communication
- remaining problems:
  - not always possible
  - replay attack

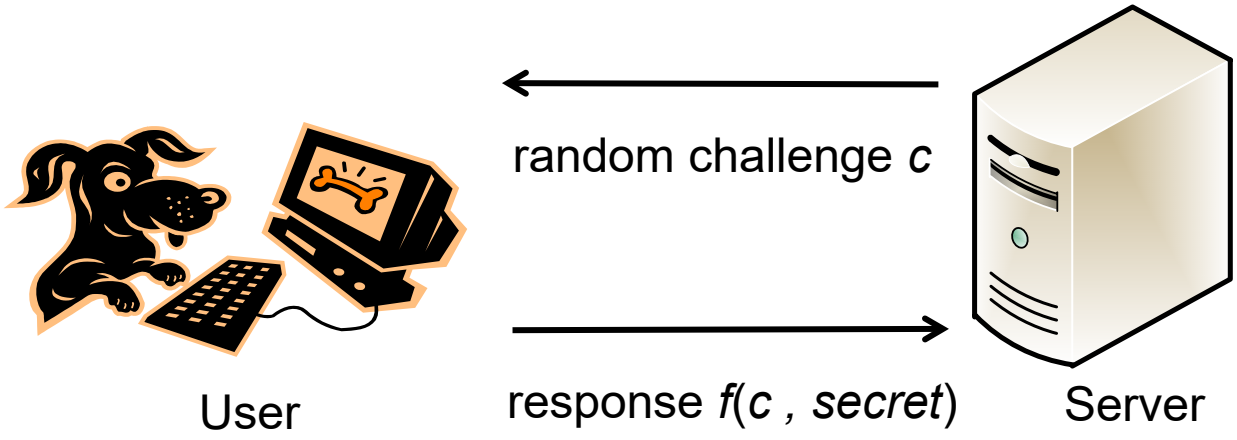
Password based authentication

- **security problem – replay attack**



Password based authentication

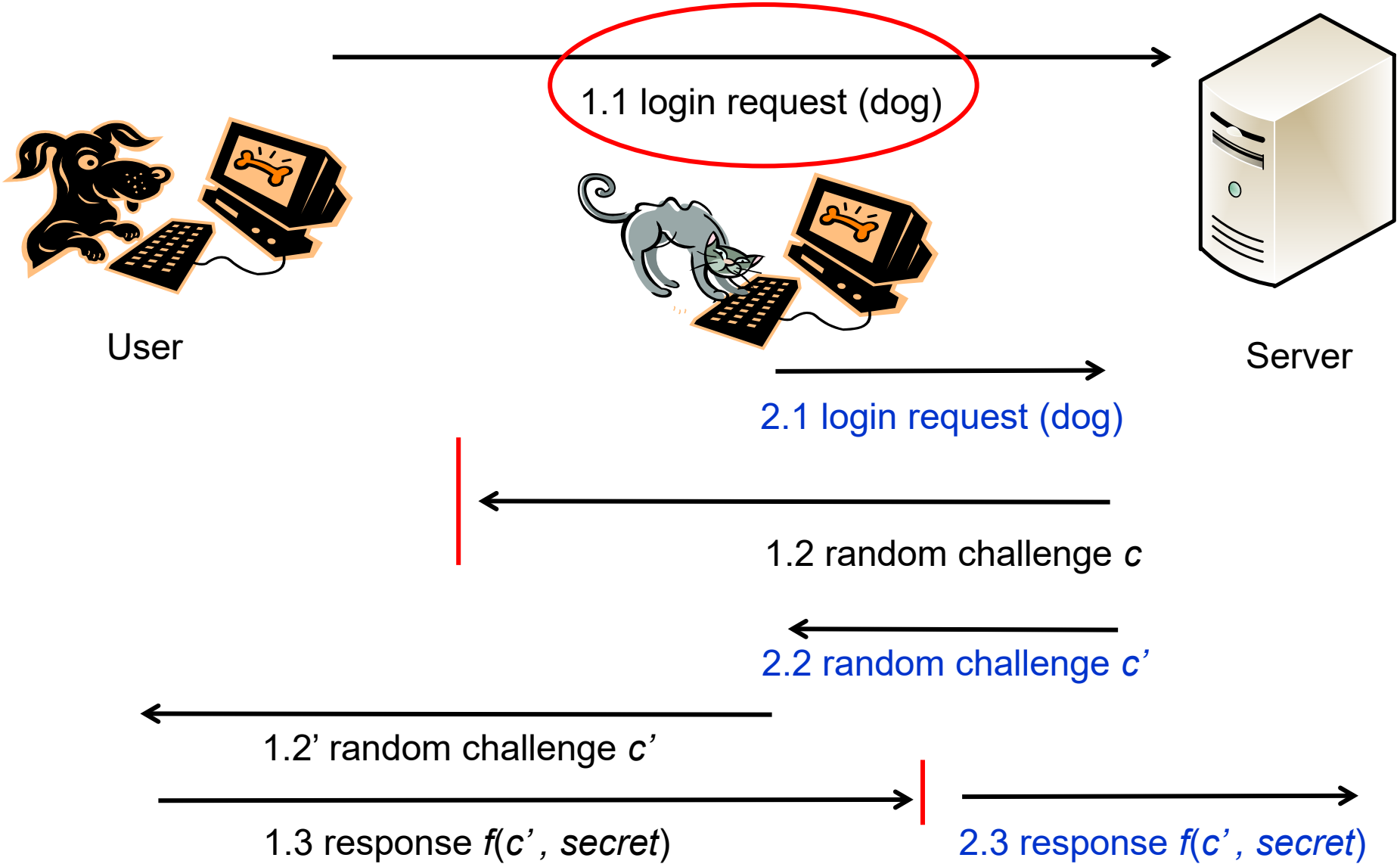
- **security problem – replay attack**
- **possible solution: challenge-response protocol**



  - tries to ensure *freshness*
- remaining problems:
  - Man-in-the-middle attacks
  - parallel protocol runs

Password based authentication

- **security problem – MITM / parallel protocol runs**

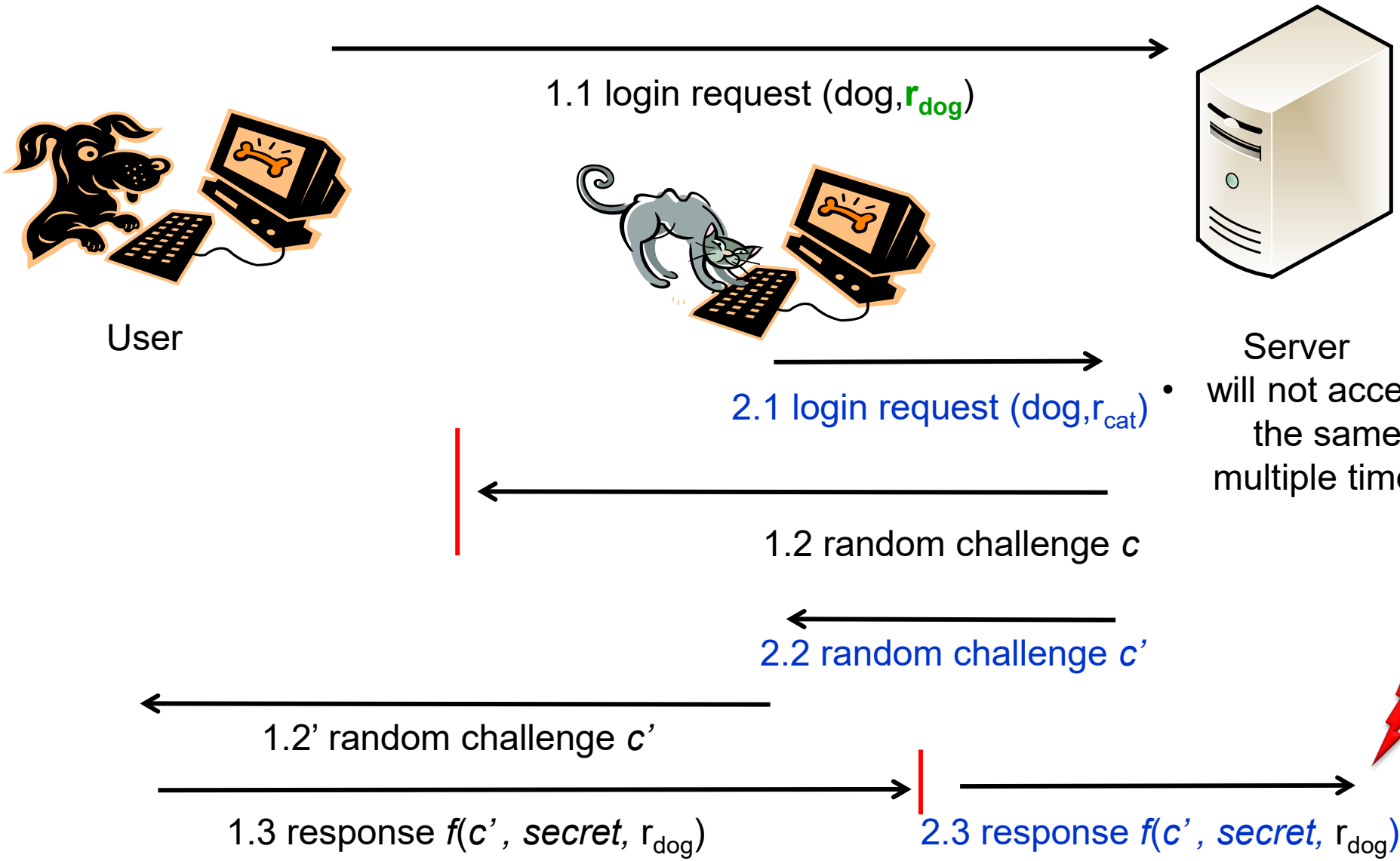


Password based authentication

- **security problem – MITM / parallel protocol runs**
- **possible solutions:**
  - disallow parallel login protocol runs for the same user
  - make protocol runs distinguishable

Password based authentication

- **possible solution: distinguishable protocol runs**



Password based authentication

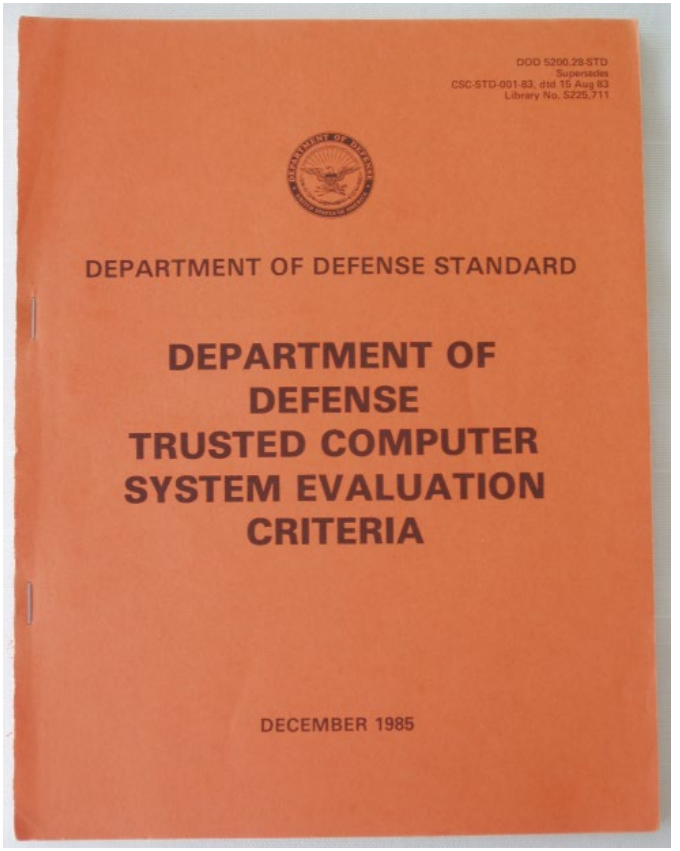
- **security problem – MITM / parallel protocol runs**
- **possible solutions:**
  - disallow parallel login protocol runs for the same user
  - make protocol runs distinguishable
- remaining security problems:
  - …

  (ok I will stop here – if you are interested in many more problems / solutions I recommend: Colin Boyd, Anish Mathuria: “Protocols for Authentication and Key Establishment”, Springer, 2003.)

Password based authentication

- **(non protocol related) security problems:**
  - phising, i.e. faked UI for entering secret information
  - today: mostly Internet based attacks
  - but: local attacks possible as well
    - faked login / lock screen
    - solution: “trusted path” / Secure Attention Key

      3.2.2.1.1 The TCB [Trusted Computing Base] shall support a trusted communication path between itself and user for initial login and authentication. Communications via this path shall be initiated exclusively by a user.
      [Department of Defense: “Trusted Computer System Evaluation Criteria”, CSC-STD-001-83, 15. August 1983 – called “Orange Book”]
    - well known implementations:
      - Windows: Ctrl+Alt+Del
      - Linux: Ctrl+Alt+Pause
        - could be freely chosen in principle



[http://en.wikipedia.org/wiki/File:Orange-book-small.PNG]

One time password

- One Time Password
  - only used to authenticate a single transation
- Advantage
  - abuse of OTP becomes harder for the attacker
- Implementations
  - list of OTPs
    - known from online banking: TAN, iTAN
  - on the fly generated and transmitted over a second channel
    - mTAN
  - time-synchronized (hardware) tokens:
    - token knows a secret *s*
    - OTP= *f*(s,time)
  - hash chain based

One time password

- OTP Implementations
  - hash chain based
    - Leslie Lamport: “Password Authentication with Insecure Communication”
    - users generates hash chain:
      - $h^n(\ldots h^3(h^2(h^1(\text{password}))))$
    - users sends $\boldsymbol{h^n()}$ as his “password” during register procedure
    - next login user sends $\boldsymbol{h^{n-1}()}$
    - server verifies: h($\boldsymbol{h^{n-1}()}$) = $\boldsymbol{h^n()}$
    - server now stores: $\boldsymbol{h^{n-1}()}$

Brute-Force im Bereich Paßwörter

Praxis

Brute-Force-Werzeuge—Burp

Burp Suite (https://portswigger.net/burp/)

- Java-basiert
- sieht sich selbst als Test-Plattform für Web-basierte Anwendungen
- ➔ kann als Brute-Force-Werkzeug verwendet werden, kann aber wesentlich mehr

- arbeitet als Proxy
- ➔ Analyse / Veränderung des Datenverkehrs zwischen Client (Browser) und (Web-)Server
  - Crawler
  - Schwachstellen-Scanner: automatisierte Tests auf bekannte Schwachstellen
  - Skript gesteuerte Anfrage-Generierung basierend auf abgefangenen Anfragen

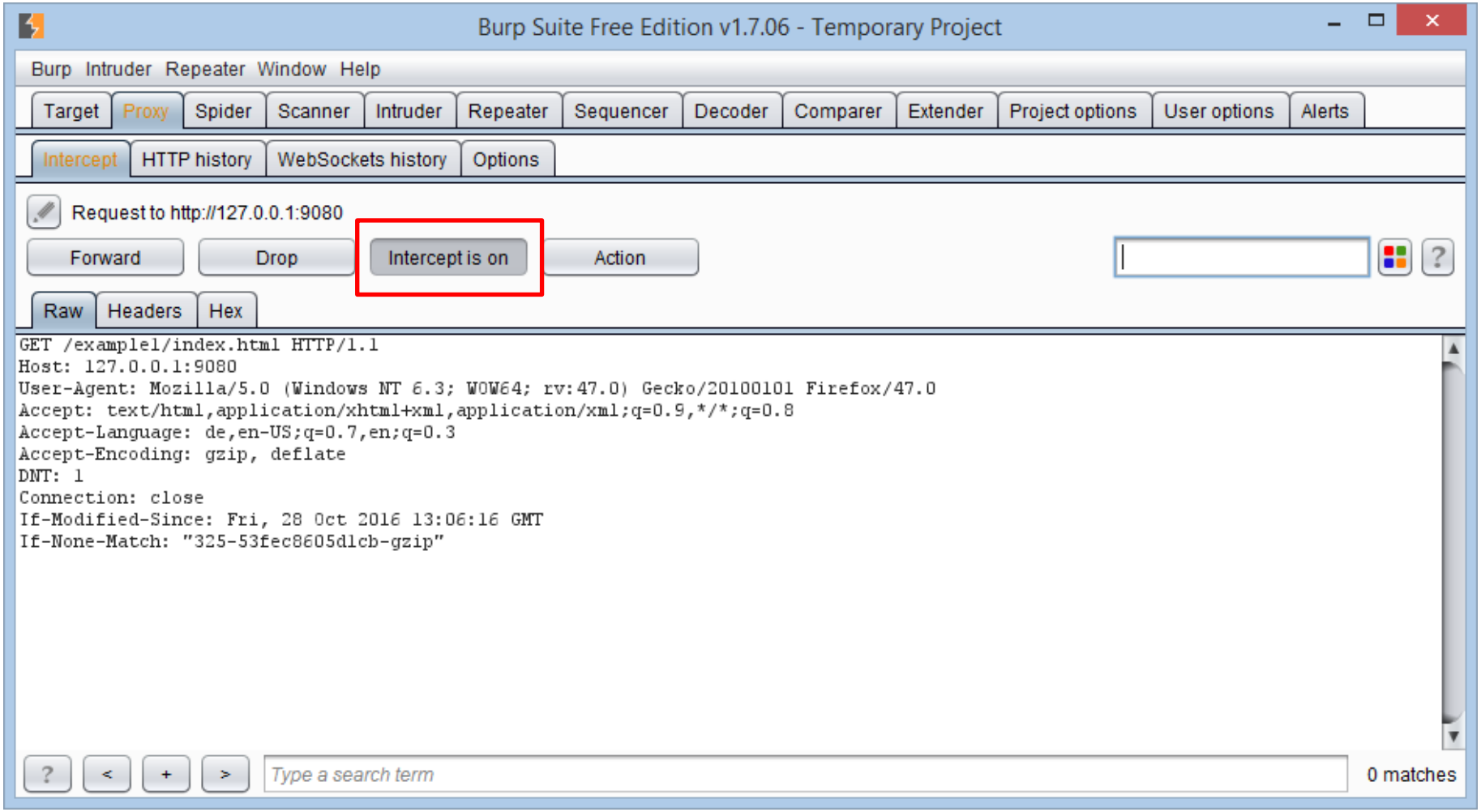
- kommerzielles Produkt
- eingeschränkte, kostenfreie Version erhältlich
  - kein Speichern möglich
  - Brute-Force: künstliche Verzögerung

Burp Anwendungsbeispiel— Bruteforce-Paßwort-Angriff

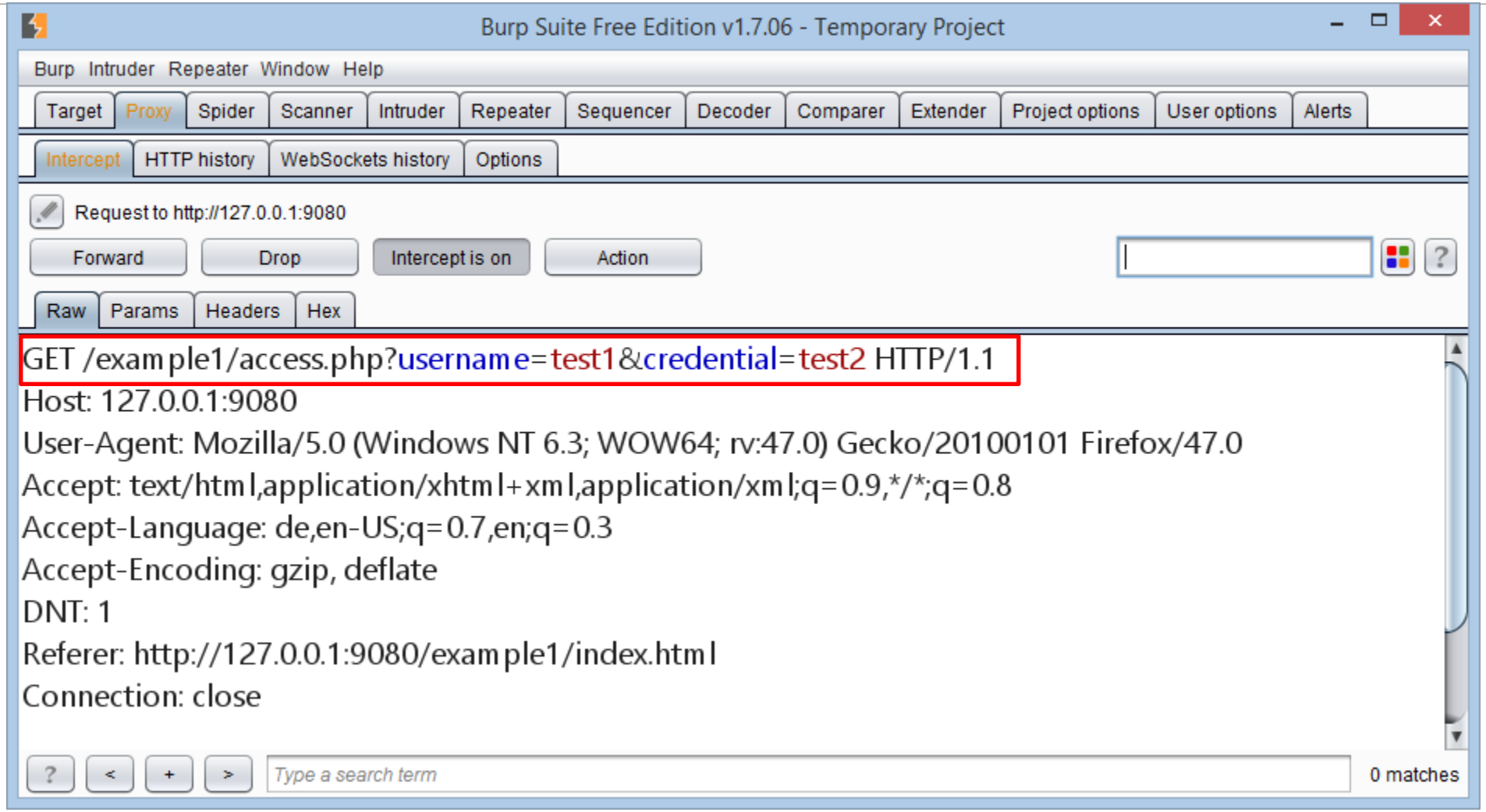
Schritte:

- Login Seite aufrufen
  - hilfreiches Firefox-Addon: Foxy Proxy
- relevante Parameter im HTTP-Datenverkehr identifizieren
- automatisierte Anfragen Durchführen
  - Word-Listen
  - Paßwort-Generierung

Request Interception

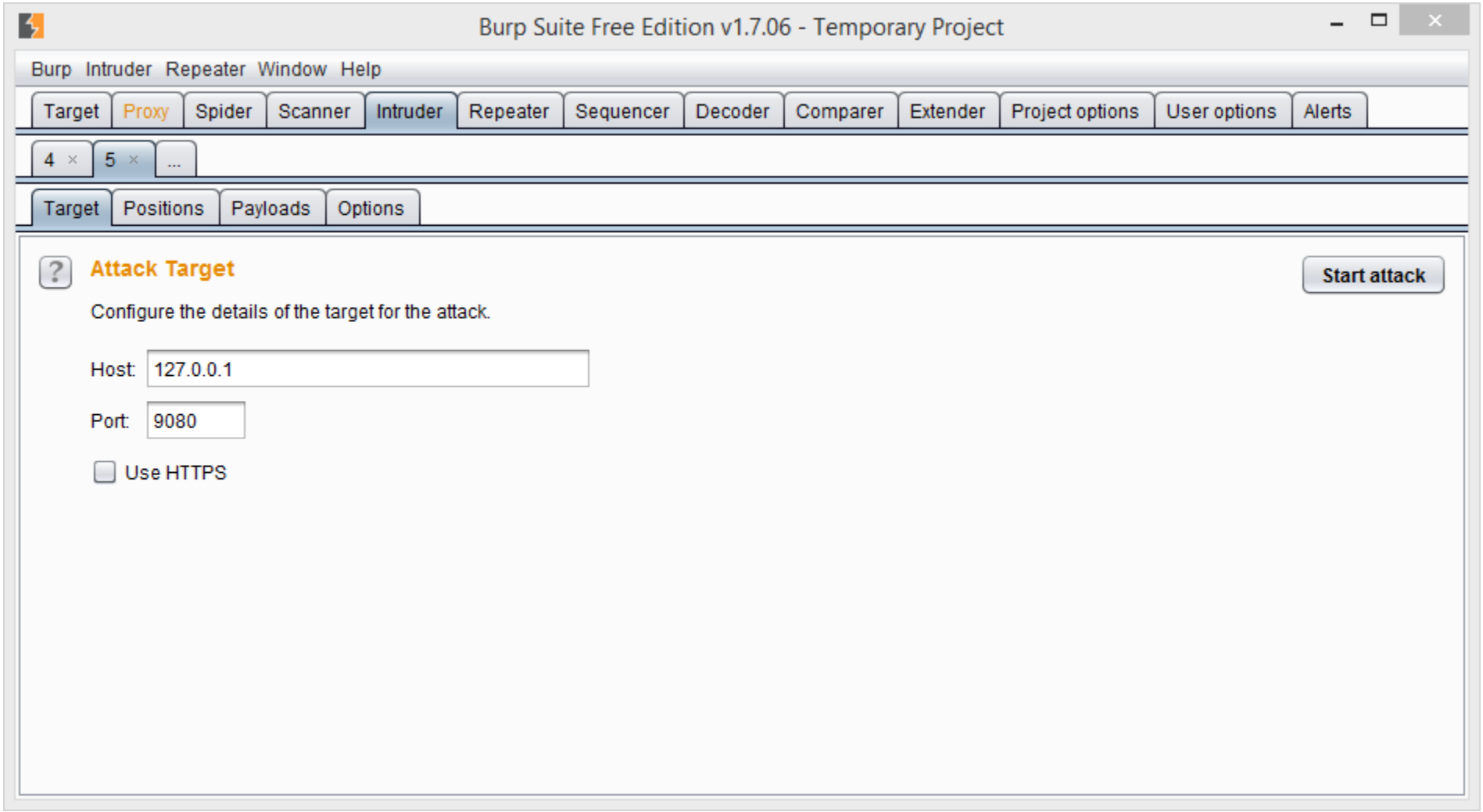


Request Interception



Action → Send to Intruder

Request Manipulation



zu testende Parameter und Parameterwerte festlegen

ZAP: OWASP Zed Attack Proxy Project

- OWASP: Open Web Application Security Project
  - https://www.owasp.org
  - Gemeinschaft von an Sicherheit von Web-Anwendungen Interessierter
  - stellt Anleitungen, Dokumente, Werkzeuge, Praxisberichte etc. zur freien Verfügung

- Proxy
- Java-basiert
- Open Source
- ähnlicher Funktionsumfang wie Burp
- Werkzeug für Brute-Force-Angriffe: Fuzzer

THC Hydra

- https://github.com/vanhauser-thc/thc-hydra
- Paßwort-Brute-Force-Werkzeug
  - online Paßwort-Brechen
- Kommandozeilen-Werkzeug
  - grafische Benutzungsoberfläche vorhanden
- Vielzahl an unterstützten Protokollen
  - 51 eingebaut
  - erweiterbar
- programmiert in C
- Open Source

Hydra Beispiel

```
>hydra -U http-get-form
```

Hilfstext zur Benutzung von `hydra` im Falle von HTTP-GET übertragenen Login-Formularen

Syntax: <url>:<form parameters>:<condition string>

Examples:

```
"/login.php:user=^USER^&pass=^PASS^:incorrect"

>hydra -L usernames.txt -p test 192.168.1.10 http-get-form
   "/example1/access.php:username=^USER^&credential=^PASS^:User"

>hydra -l root -P passwords.txt 192.168.1.10 http-get-form
   "/example1/access.php:username=^USER^&credential=^PASS^:fail"
```

John the Ripper

- http://www.openwall.com/john/
- Zweck: offline Paßwort brechen
  - Wörterbuch-Angriffe
  - Regel-basierte Paßwortgenerierung
    - eingebaute Heuristiken
    - (mächtige) Beschreibungssprache für eigene Regeln
  - frei programmierbare Paßwortgenerierung
    - C ähnliche Sprache

- drei Versionen
  - kommerzielle Pro-Version
  - Open Source Version
  - Community-Version – enthält viele Zusatzpakete, aber weniger getestet

Online Quellen

- Milliarden von Paßwortdatensätzen sind im Internet auffindbar

- Vielzahl an Web-Seiten / Online-Diensten, die Paßwörter zu verschiedenen Hash-basierten Authentifizierungsmechanismen liefern
  - https://crackstation.net/
  - https://hashcrack.org/

- Rainbow-Tables zum Download
  - http://project-rainbowcrack.com/table.htm

- Überprüfung bzgl. Sicherheitsleaks
  - https://sec.hpi.uni-potsdam.de/leak-checker/search
  - https://haveibeenpwned.com/

$h(x) := (x \cdot 7807) \bmod 16157$
$f(x) := x \bmod 9000 + 1000$

andi:11500