CE0973a - Issues in Network Security
11: Authentication, Passwords

James A Sutherland

Abertay University

Monday, 21st March 2016

Password Hashing

- Early ‘encryption’: instead of storing the password as-is, mangle it.
- Unix `crypt` function encrypted the password, using itself as a key.
- That was too fast. Enter DES, with salt.
- 8 ASCII (7 bit) characters: 56 bits.
- Slightly modified DES, ‘perturbed’ by 12 bit salt value.
- Originally, `/etc/passwd` held password, world-readable![1]

[1] Since mid-80s, `/etc/shadow` holds these instead, root-only

Password Hashing

- Early ‘encryption’: instead of storing the password as-is, mangle it.
- Unix `crypt` function encrypted the password, using itself as a key.
- That was too fast. Enter DES, with salt.
- 8 ASCII (7 bit) characters: 56 bits.
- Slightly modified DES, ‘perturbed’ by 12 bit salt value.
- Originally, `/etc/passwd` held password, world-readable![1]

[1] Since mid-80s, `/etc/shadow` holds these instead, root-only

Password Hashing

- Early ‘encryption’: instead of storing the password as-is, mangle it.
- Unix `crypt` function encrypted the password, using itself as a key.
- That was too fast. Enter DES, with salt.
- 8 ASCII (7 bit) characters: 56 bits.
- Slightly modified DES, ‘perturbed’ by 12 bit salt value.
- Originally, `/etc/passwd` held password, world-readable![1]

[1] Since mid-80s, `/etc/shadow` holds these instead, root-only

Password Hashing

- Early 'encryption': instead of storing the password as-is, mangle it.
- Unix `crypt` function encrypted the password, using itself as a key.
- That was too fast. Enter DES, with salt.
- 8 ASCII (7 bit) characters: 56 bits.
- Slightly modified DES, 'perturbed' by 12 bit salt value.
- Originally, `/etc/passwd` held password, world-readable![1]

[1] Since mid-80s, `/etc/shadow` holds these instead, root-only

Password Hashing

- Early ‘encryption’: instead of storing the password as-is, mangle it.
- Unix `crypt` function encrypted the password, using itself as a key.
- That was too fast. Enter DES, with salt.
- 8 ASCII (7 bit) characters: 56 bits.
- Slightly modified DES, ‘perturbed’ by 12 bit salt value.
- Originally, `/etc/passwd` held password, world-readable![1]

[1] Since mid-80s, `/etc/shadow` holds these instead, root-only

Password Hashing

- Early 'encryption': instead of storing the password as-is, mangle it.
- Unix `crypt` function encrypted the password, using itself as a key.
- That was too fast. Enter DES, with salt.
- 8 ASCII (7 bit) characters: 56 bits.
- Slightly modified DES, 'perturbed' by 12 bit salt value.
- Originally, `/etc/passwd` held password, world-readable![1]

[1] Since mid-80s, `/etc/shadow` holds these instead, root-only

Traditional crypt limitations

- Only eight characters
- ASCII only: no accents, pound signs. . .
- Total 56 bits
- Skewed, though, very few NULL keys on keyboards

Traditional `crypt` limitations

- Only eight characters
- ASCII only: no accents, pound signs. . .
- Total 56 bits
- Skewed, though, very few NULL keys on keyboards

Traditional `crypt` limitations

- Only eight characters
- ASCII only: no accents, pound signs...
- Total 56 bits
- Skewed, though, very few NULL keys on keyboards

Traditional crypt limitations

- Only eight characters
- ASCII only: no accents, pound signs. . .
- Total 56 bits
- Skewed, though, very few NULL keys on keyboards

Windows

- Originally *optional* login (bypass by hitting Escape!)
- Then Windows LAN Manager (DOS, OS/2)
- ‘LM hash’
- Case insensitive ASCII subset
- Split in two 7 character pieces
- Brute force in hours, rainbow table in seconds
- No salt (and network use meant replay attacks, pass the hash etc)
- DES based, but 7 not 8 characters, no salt: much weaker than Unix

Windows

- Originally *optional* login (bypass by hitting Escape!)
- Then Windows LAN Manager (DOS, OS/2)
- 'LM hash'
- Case insensitive ASCII subset
- Split in two 7 character pieces
- Brute force in hours, rainbow table in seconds
- No salt (and network use meant replay attacks, pass the hash etc)
- DES based, but 7 not 8 characters, no salt: much weaker than Unix

Windows

- Originally *optional* login (bypass by hitting Escape!)
- Then Windows LAN Manager (DOS, OS/2)
- ‘LM hash’
- Case insensitive ASCII subset
- Split in two 7 character pieces
- Brute force in hours, rainbow table in seconds
- No salt (and network use meant replay attacks, pass the hash etc)
- DES based, but 7 not 8 characters, no salt: much weaker than Unix

Windows

- Originally *optional* login (bypass by hitting Escape!)
- Then Windows LAN Manager (DOS, OS/2)
- 'LM hash'
- Case insensitive ASCII subset
- Split in two 7 character pieces
- Brute force in hours, rainbow table in seconds
- No salt (and network use meant replay attacks, pass the hash etc)
- DES based, but 7 not 8 characters, no salt: much weaker than Unix

Windows

- Originally *optional* login (bypass by hitting Escape!)
- Then Windows LAN Manager (DOS, OS/2)
- 'LM hash'
- Case insensitive ASCII subset
- Split in two 7 character pieces
- Brute force in hours, rainbow table in seconds
- No salt (and network use meant replay attacks, pass the hash etc)
- DES based, but 7 not 8 characters, no salt: much weaker than Unix

Windows

- Originally *optional* login (bypass by hitting Escape!)
- Then Windows LAN Manager (DOS, OS/2)
- 'LM hash'
- Case insensitive ASCII subset
- Split in two 7 character pieces
- Brute force in hours, rainbow table in seconds
- No salt (and network use meant replay attacks, pass the hash etc)
- DES based, but 7 not 8 characters, no salt: much weaker than Unix

Windows

- Originally *optional* login (bypass by hitting Escape!)
- Then Windows LAN Manager (DOS, OS/2)
- 'LM hash'
- Case insensitive ASCII subset
- Split in two 7 character pieces
- Brute force in hours, rainbow table in seconds
- No salt (and network use meant replay attacks, pass the hash etc)
- DES based, but 7 not 8 characters, no salt: much weaker than Unix

Windows

- Originally *optional* login (bypass by hitting Escape!)
- Then Windows LAN Manager (DOS, OS/2)
- 'LM hash'
- Case insensitive ASCII subset
- Split in two 7 character pieces
- Brute force in hours, rainbow table in seconds
- No salt (and network use meant replay attacks, pass the hash etc)
- DES based, but 7 not 8 characters, no salt: much weaker than Unix

Windows, mk II

- Negotiates, like SSL: ‘I speak X,Y’, ‘I only speak X’
- Introduced second hash, NT hash (MD4 based – broken predecessor to MD5)
- Backwards compatibility meant both used in parallel for years
- NTLMv1 protocol introduced challenge-response (avoid replay attacks)
- Still used DES though
- Windows NT 4 SP4 (and Win2k) introduced NTLMv2, HMAC-MD5 based

Windows, mk II

- Negotiates, like SSL: ‘I speak X,Y’, ‘I only speak X’
- Introduced second hash, NT hash (MD4 based – broken predecessor to MD5)
- Backwards compatibility meant both used in parallel for years
- NTLMv1 protocol introduced challenge-response (avoid replay attacks)
- Still used DES though
- Windows NT 4 SP4 (and Win2k) introduced NTLMv2, HMAC-MD5 based

Windows, mk II

- Negotiates, like SSL: ‘I speak X,Y’, ‘I only speak X’
- Introduced second hash, NT hash (MD4 based – broken predecessor to MD5)
- Backwards compatibility meant both used in parallel for years
- NTLMv1 protocol introduced challenge-response (avoid replay attacks)
- Still used DES though
- Windows NT 4 SP4 (and Win2k) introduced NTLMv2, HMAC-MD5 based

Windows, mk II

- Negotiates, like SSL: ‘I speak X,Y’, ‘I only speak X’
- Introduced second hash, NT hash (MD4 based – broken predecessor to MD5)
- Backwards compatibility meant both used in parallel for years
- NTLMv1 protocol introduced challenge-response (avoid replay attacks)
- Still used DES though
- Windows NT 4 SP4 (and Win2k) introduced NTLMv2, HMAC-MD5 based

Windows, mk II

- Negotiates, like SSL: ‘I speak X,Y’, ‘I only speak X’
- Introduced second hash, NT hash (MD4 based – broken predecessor to MD5)
- Backwards compatibility meant both used in parallel for years
- NTLMv1 protocol introduced challenge-response (avoid replay attacks)
- Still used DES though
- Windows NT 4 SP4 (and Win2k) introduced NTLMv2, HMAC-MD5 based

Windows, mk II

- Negotiates, like SSL: ‘I speak X,Y’, ‘I only speak X’
- Introduced second hash, NT hash (MD4 based – broken predecessor to MD5)
- Backwards compatibility meant both used in parallel for years
- NTLMv1 protocol introduced challenge-response (avoid replay attacks)
- Still used DES though
- Windows NT 4 SP4 (and Win2k) introduced NTLMv2, HMAC-MD5 based

Windows Today

- Windows XP pre-SP3 affected by US encryption limits
- Windows Vista rejects LM auth, no LM hash[2]
- “Weak nonce” problem documented in 2010, fix MS10-012[3]
- 14 year old issue, from Windows NT 3.1 (1993) to Windows 7
- Bad random number generator – easily done
- Finally moving to Kerberos, a 1980s auth system from MIT

[2] By default. Just turn the security off if it gets in the way…

[3] http://www.ampliasecurity.com/research/NTLMWeakNonce-bh2010-usa-ampliasecurity.pdf

Windows Today

- Windows XP pre-SP3 affected by US encryption limits
- Windows Vista rejects LM auth, no LM hash[2]
- “Weak nonce” problem documented in 2010, fix MS10-012[3]
- 14 year old issue, from Windows NT 3.1 (1993) to Windows 7
- Bad random number generator – easily done
- Finally moving to Kerberos, a 1980s auth system from MIT

[2] By default. Just turn the security off if it gets in the way…

[3] http://www.ampliasecurity.com/research/NTLMWeakNonce-bh2010-usa-ampliasecurity.pdf

Windows Today

- Windows XP pre-SP3 affected by US encryption limits
- Windows Vista rejects LM auth, no LM hash[2]
- "Weak nonce" problem documented in 2010, fix MS10-012[3]
- 14 year old issue, from Windows NT 3.1 (1993) to Windows 7
- Bad random number generator – easily done
- Finally moving to Kerberos, a 1980s auth system from MIT

[2] By default. Just turn the security off if it gets in the way. . .

[3] http://www.ampliasecurity.com/research/NTLMWeakNonce-bh2010-usa-ampliasecurity.pdf

Windows Today

- Windows XP pre-SP3 affected by US encryption limits
- Windows Vista rejects LM auth, no LM hash[2]
- "Weak nonce" problem documented in 2010, fix MS10-012[3]
- 14 year old issue, from Windows NT 3.1 (1993) to Windows 7
- Bad random number generator – easily done
- Finally moving to Kerberos, a 1980s auth system from MIT

[2] By default. Just turn the security off if it gets in the way...

[3] http://www.ampliasecurity.com/research/NTLMWeakNonce-bh2010-usa-ampliasecurity.pdf

Windows Today

- Windows XP pre-SP3 affected by US encryption limits
- Windows Vista rejects LM auth, no LM hash[2]
- "Weak nonce" problem documented in 2010, fix MS10-012[3]
- 14 year old issue, from Windows NT 3.1 (1993) to Windows 7
- Bad random number generator – easily done
- Finally moving to Kerberos, a 1980s auth system from MIT

[2] By default. Just turn the security off if it gets in the way...

[3] http://www.ampliasecurity.com/research/NTLMWeakNonce-bh2010-usa-ampliasecurity.pdf

Windows Today

- Windows XP pre-SP3 affected by US encryption limits
- Windows Vista rejects LM auth, no LM hash[2]
- “Weak nonce” problem documented in 2010, fix MS10-012[3]
- 14 year old issue, from Windows NT 3.1 (1993) to Windows 7
- Bad random number generator – easily done
- Finally moving to Kerberos, a 1980s auth system from MIT

[2] By default. Just turn the security off if it gets in the way...

[3] http://www.ampliasecurity.com/research/NTLMWeakNonce-bh2010-usa-ampliasecurity.pdf

Other Systems

- Windows Active Directory (replaced NT Domains)
- RADIUS (Eduroam)
- LDAP (Apple OpenDirectory)
- NIS, NIS+
- Novell NDS

Other Systems

- Windows Active Directory (replaced NT Domains)
- RADIUS (Eduroam)
- LDAP (Apple OpenDirectory)
- NIS, NIS+
- Novell NDS

Other Systems

- Windows Active Directory (replaced NT Domains)
- RADIUS (Eduroam)
- LDAP (Apple OpenDirectory)
- NIS, NIS+
- Novell NDS

Other Systems

- Windows Active Directory (replaced NT Domains)
- RADIUS (Eduroam)
- LDAP (Apple OpenDirectory)
- NIS, NIS+
- Novell NDS

Other Systems

- Windows Active Directory (replaced NT Domains)
- RADIUS (Eduroam)
- LDAP (Apple OpenDirectory)
- NIS, NIS+
- Novell NDS

User Accounts in General

- Most systems replicate user data across multiple servers for performance, availability.
- Makes changes problematic (delayed sync, inconsistency issues)
- Also security challenge: syncing securely, trust

User Accounts in General

- Most systems replicate user data across multiple servers for performance, availability.
- Makes changes problematic (delayed sync, inconsistency issues)
- Also security challenge: syncing securely, trust

User Accounts in General

- Most systems replicate user data across multiple servers for performance, availability.
- Makes changes problematic (delayed sync, inconsistency issues)
- Also security challenge: syncing securely, trust

Salting Passwords

- Why salt passwords?
- Otherwise, if Fred and Jim both have a password $0x12345678$, they have the same password.
- Complicates brute-forcing: n different encryptions for each password.
- Pre-computed dictionary table also much harder.
- So, general principle: shove randomness or something user-specific in.

Salting Passwords

- Why salt passwords?
- Otherwise, if Fred and Jim both have a password $0x12345678$, they have the same password.
- Complicates brute-forcing: n different encryptions for each password.
- Pre-computed dictionary table also much harder.
- So, general principle: shove randomness or something user-specific in.

Salting Passwords

- Why salt passwords?
- Otherwise, if Fred and Jim both have a password $0x12345678$, they have the same password.
- Complicates brute-forcing: n different encryptions for each password.
- Pre-computed dictionary table also much harder.
- So, general principle: shove randomness or something user-specific in.

Salting Passwords

- Why salt passwords?
- Otherwise, if Fred and Jim both have a password 0x12345678, they have the same password.
- Complicates brute-forcing: n different encryptions for each password.
- Pre-computed dictionary table also much harder.
- So, general principle: shove randomness or something user-specific in.

Salting Passwords

- Why salt passwords?
- Otherwise, if Fred and Jim both have a password 0x12345678, they have the same password.
- Complicates brute-forcing: n different encryptions for each password.
- Pre-computed dictionary table also much harder.
- So, general principle: shove randomness or something user-specific in.

Passwords in General

- Salt your hashes
- Use *strong* random numbers to guard against replays
- Sign traffic to guard against tampering (see MS SQL Ethernet exploit)
- Don't DIY: MS blew $m botching it in-house before Kerberos!
- (NTLM still used if not in domain though)
- Never accept the stored form, otherwise it becomes plaintext-equivalent!

Passwords in General

- Salt your hashes
- Use *strong* random numbers to guard against replays
- Sign traffic to guard against tampering (see MS SQL Ethernet exploit)
- Don't DIY: MS blew $m botching it in-house before Kerberos!
- (NTLM still used if not in domain though)
- Never accept the stored form, otherwise it becomes plaintext-equivalent!

Passwords in General

- Salt your hashes
- Use *strong* random numbers to guard against replays
- Sign traffic to guard against tampering (see MS SQL Ethernet exploit)
- Don't DIY: MS blew $m botching it in-house before Kerberos!
- (NTLM still used if not in domain though)
- Never accept the stored form, otherwise it becomes plaintext-equivalent!

Passwords in General

- Salt your hashes
- Use *strong* random numbers to guard against replays
- Sign traffic to guard against tampering (see MS SQL Ethernet exploit)
- Don't DIY: MS blew $m botching it in-house before Kerberos!
- (NTLM still used if not in domain though)
- Never accept the stored form, otherwise it becomes plaintext-equivalent!

Passwords in General

- Salt your hashes
- Use *strong* random numbers to guard against replays
- Sign traffic to guard against tampering (see MS SQL Ethernet exploit)
- Don't DIY: MS blew $m botching it in-house before Kerberos!
- (NTLM still used if not in domain though)
- Never accept the stored form, otherwise it becomes plaintext-equivalent!

Passwords in General

- Salt your hashes
- Use *strong* random numbers to guard against replays
- Sign traffic to guard against tampering (see MS SQL Ethernet exploit)
- Don't DIY: MS blew $m botching it in-house before Kerberos!
- (NTLM still used if not in domain though)
- Never accept the stored form, otherwise it becomes plaintext-equivalent!

Network Authentication

- SSL client certificates
  - Basically like server ones (but easier to issue)
- IP based
  - Just what it says: 192.168.*.* can print
- DNS based
  - Slightly more interesting, spoofable: *.uad.ac.uk
- Passwords and password protocols
- Central password servers: RADIUS, LDAP, MS AD

Network Authentication

- SSL client certificates
  - Basically like server ones (but easier to issue)
- IP based
  - Just what it says: 192.168.*.* can print
- DNS based
  - Slightly more interesting, spoofable: *.uad.ac.uk
- Passwords and password protocols
- Central password servers: RADIUS, LDAP, MS AD

Network Authentication

- SSL client certificates
  - Basically like server ones (but easier to issue)
- IP based
  - Just what it says: 192.168.*.* can print
- DNS based
  - Slightly more interesting, spoofable: *.uad.ac.uk
- Passwords and password protocols
- Central password servers: RADIUS, LDAP, MS AD

Network Authentication

- SSL client certificates
  - Basically like server ones (but easier to issue)
- IP based
  - Just what it says: 192.168.*.* can print
- DNS based
  - Slightly more interesting, spoofable: *.uad.ac.uk
- Passwords and password protocols
- Central password servers: RADIUS, LDAP, MS AD

Network Authentication

- SSL client certificates
  - Basically like server ones (but easier to issue)
- IP based
  - Just what it says: 192.168.*.* can print
- DNS based
  - Slightly more interesting, spoofable: *.uad.ac.uk
- Passwords and password protocols
- Central password servers: RADIUS, LDAP, MS AD

Network Authentication

- SSL client certificates
  - Basically like server ones (but easier to issue)
- IP based
  - Just what it says: 192.168.*.* can print
- DNS based
  - Slightly more interesting, spoofable: *.uad.ac.uk
- Passwords and password protocols
- Central password servers: RADIUS, LDAP, MS AD

Network Authentication

- SSL client certificates
  - Basically like server ones (but easier to issue)
- IP based
  - Just what it says: 192.168.*.* can print
- DNS based
  - Slightly more interesting, spoofable: *.uad.ac.uk
- Passwords and password protocols
- Central password servers: RADIUS, LDAP, MS AD

Network Authentication

- SSL client certificates
  - Basically like server ones (but easier to issue)
- IP based
  - Just what it says: 192.168.*.* can print
- DNS based
  - Slightly more interesting, spoofable: *.uad.ac.uk
- Passwords and password protocols
- Central password servers: RADIUS, LDAP, MS AD

Password Protocols

- Plain text
  - Surprisingly popular especially over SSL, not too insecure there
- Challenge response, e.g. MSCHAP, HTTP Digest
  - Clever challenge response: sends random numbers, requires hash of that+password
  - Downside: requires specific form of password (Digest) or plaintext (MSCHAP)
- Kerberos
  - Trade password for a token to use elsewhere
  - Yes, Windows usually keeps your password in plaintext to reuse![4]

[4] http://blog.opensecurityresearch.com/2012/06/using-mimikatz-to-dump-passwords.html

Password Protocols

- Plain text
  - Surprisingly popular especially over SSL, not too insecure there
- Challenge response, e.g. MSCHAP, HTTP Digest
  - Clever challenge response: sends random numbers, requires hash of that+password
  - Downside: requires specific form of password (Digest) or plaintext (MSCHAP)
- Kerberos
  - Trade password for a token to use elsewhere
  - Yes, Windows usually keeps your password in plaintext to reuse![4]

[4] http://blog.opensecurityresearch.com/2012/06/using-mimikatz-to-dump-passwords.html

Password Protocols

- Plain text
  - Surprisingly popular especially over SSL, not too insecure there
- Challenge response, e.g. MSCHAP, HTTP Digest
  - Clever challenge response: sends random numbers, requires hash of that+password
  - Downside: requires specific form of password (Digest) or plaintext (MSCHAP)
- Kerberos
  - Trade password for a token to use elsewhere
  - Yes, Windows usually keeps your password in plaintext to reuse![4]

[4] http://blog.opensecurityresearch.com/2012/06/using-mimikatz-to-dump-passwords.html

Password Protocols

- Plain text
  - Surprisingly popular especially over SSL, not too insecure there
- Challenge response, e.g. MSCHAP, HTTP Digest
  - Clever challenge response: sends random numbers, requires hash of that+password
  - Downside: requires specific form of password (Digest) or plaintext (MSCHAP)
- Kerberos
  - Trade password for a token to use elsewhere
  - Yes, Windows usually keeps your password in plaintext to reuse![4]

[4] http://blog.opensecurityresearch.com/2012/06/using-mimikatz-to-dump-passwords.html

Password Protocols

- Plain text
  - Surprisingly popular especially over SSL, not too insecure there
- Challenge response, e.g. MSCHAP, HTTP Digest
  - Clever challenge response: sends random numbers, requires hash of that+password
  - Downside: requires specific form of password (Digest) or plaintext (MSCHAP)
- Kerberos
  - Trade password for a token to use elsewhere
  - Yes, Windows usually keeps your password in plaintext to reuse![4]

[4] http://blog.opensecurityresearch.com/2012/06/using-mimikatz-to-dump-passwords.html

Password Protocols

- Plain text
  - Surprisingly popular especially over SSL, not too insecure there
- Challenge response, e.g. MSCHAP, HTTP Digest
  - Clever challenge response: sends random numbers, requires hash of that+password
  - Downside: requires specific form of password (Digest) or plaintext (MSCHAP)
- Kerberos
  - Trade password for a token to use elsewhere
  - Yes, Windows usually keeps your password in plaintext to reuse![4]

[4] http://blog.opensecurityresearch.com/2012/06/using-mimikatz-to-dump-passwords.html

Password Protocols

- Plain text
  - Surprisingly popular especially over SSL, not too insecure there
- Challenge response, e.g. MSCHAP, HTTP Digest
  - Clever challenge response: sends random numbers, requires hash of that+password
  - Downside: requires specific form of password (Digest) or plaintext (MSCHAP)
- Kerberos
  - Trade password for a token to use elsewhere
  - Yes, Windows usually keeps your password in plaintext to reuse![4]

[4] http://blog.opensecurityresearch.com/2012/06/using-mimikatz-to-dump-passwords.html

Password Protocols

- Plain text
  - Surprisingly popular especially over SSL, not too insecure there
- Challenge response, e.g. MSCHAP, HTTP Digest
  - Clever challenge response: sends random numbers, requires hash of that+password
  - Downside: requires specific form of password (Digest) or plaintext (MSCHAP)
- Kerberos
  - Trade password for a token to use elsewhere
  - Yes, Windows usually keeps your password in plaintext to reuse![4]

[4] `http://blog.opensecurityresearch.com/2012/06/using-mimikatz-to-dump-passwords.html`

Bonus Content – Buffer Overflow

- When you call a function, your address gets pushed on the stack to return to
- When you allocate a small buffer, that's usually on the stack
- So, allocate a buffer then call a function . . .
- . . . going beyond the end changes the return address . . .
- . . . and now you're calling the attacker's code instead!

Bonus Content – Buffer Overflow

- When you call a function, your address gets pushed on the stack to return to
- When you allocate a small buffer, that's usually on the stack
- So, allocate a buffer then call a function . . .
- . . . going beyond the end changes the return address . . .
- . . . and now you're calling the attacker's code instead!

Bonus Content – Buffer Overflow

- When you call a function, your address gets pushed on the stack to return to
- When you allocate a small buffer, that's usually on the stack
- So, allocate a buffer then call a function . . .
- . . . going beyond the end changes the return address . . .
- . . . and now you're calling the attacker's code instead!

Bonus Content – Buffer Overflow

- When you call a function, your address gets pushed on the stack to return to
- When you allocate a small buffer, that's usually on the stack
- So, allocate a buffer then call a function . . .
- . . . going beyond the end changes the return address . . .
- . . . and now you're calling the attacker's code instead!

Bonus Content – Buffer Overflow

- When you call a function, your address gets pushed on the stack to return to
- When you allocate a small buffer, that's usually on the stack
- So, allocate a buffer then call a function ...
- ... going beyond the end changes the return address ...
- ... and now you're calling the attacker's code instead!

Buffer Overflow Variations

- Lots of variants on that vulnerability
- Return-oriented programming
- Address Space Layout Randomization mitigates
- W^X: every page *either* writable *or* executable, never both
- That causes iOS code signing and JIT conflicts
- Interpreters still vulnerable!
- Integer overflow/underflow issues too

Buffer Overflow Variations

- Lots of variants on that vulnerability
- Return-oriented programming
- Address Space Layout Randomization mitigates
- W^X: every page *either* writable *or* executable, never both
- That causes iOS code signing and JIT conflicts
- Interpreters still vulnerable!
- Integer overflow/underflow issues too

Buffer Overflow Variations

- Lots of variants on that vulnerability
- Return-oriented programming
- Address Space Layout Randomization mitigates
- W^X: every page *either* writable *or* executable, never both
- That causes iOS code signing and JIT conflicts
- Interpreters still vulnerable!
- Integer overflow/underflow issues too

Buffer Overflow Variations

- Lots of variants on that vulnerability
- Return-oriented programming
- Address Space Layout Randomization mitigates
- W^X: every page *either* writable *or* executable, never both
- That causes iOS code signing and JIT conflicts
- Interpreters still vulnerable!
- Integer overflow/underflow issues too

Buffer Overflow Variations

- Lots of variants on that vulnerability
- Return-oriented programming
- Address Space Layout Randomization mitigates
- W^X: every page *either* writable *or* executable, never both
- That causes iOS code signing and JIT conflicts
- Interpreters still vulnerable!
- Integer overflow/underflow issues too

Buffer Overflow Variations

- Lots of variants on that vulnerability
- Return-oriented programming
- Address Space Layout Randomization mitigates
- W^X: every page *either* writable *or* executable, never both
- That causes iOS code signing and JIT conflicts
- Interpreters still vulnerable!
- Integer overflow/underflow issues too

Buffer Overflow Variations

- Lots of variants on that vulnerability
- Return-oriented programming
- Address Space Layout Randomization mitigates
- W^X: every page *either* writable *or* executable, never both
- That causes iOS code signing and JIT conflicts
- Interpreters still vulnerable!
- Integer overflow/underflow issues too

Example: Mixed Overflow Exploit

- Download code, uses buffer with 20 byte header
- Send size of $0xffffffff$ $(2^{32} - 1)$ bytes
- Checks buffer size: overflows to give 19 bytes
- Allocates 19 byte buffer, proceeds to read 4Gb into it. . .

Example: Mixed Overflow Exploit

- Download code, uses buffer with 20 byte header
- Send size of $0xffffffff\,(2^{32}-1)$ bytes
- Checks buffer size: overflows to give 19 bytes
- Allocates 19 byte buffer, proceeds to read 4Gb into it. . .

Example: Mixed Overflow Exploit

- Download code, uses buffer with 20 byte header
- Send size of $0xffffffff\,(2^{32} - 1)$ bytes
- Checks buffer size: overflows to give 19 bytes
- Allocates 19 byte buffer, proceeds to read 4Gb into it. . .

Example: Mixed Overflow Exploit

- Download code, uses buffer with 20 byte header
- Send size of $0xffffffff$ $(2^{32} - 1)$ bytes
- Checks buffer size: overflows to give 19 bytes
- Allocates 19 byte buffer, proceeds to read 4Gb into it. . .

Week 11 Tasks

- Pick a password. Set *two* accounts to that.
- Dump encrypted passwords on both Win7 *and* Linux
- Try cracking both – 0phcrack[5] for Win
- Look at both encrypted forms. What can you tell?
- Also try extracting plaintext passwords on both.
- Restore Windows and Linux images before you leave!

[5] http://ophcrack.sourceforge.net

Week 11 Tasks

- Pick a password. Set *two* accounts to that.
- Dump encrypted passwords on both Win7 *and* Linux
- Try cracking both – 0phcrack[5] for Win
- Look at both encrypted forms. What can you tell?
- Also try extracting plaintext passwords on both.
- Restore Windows and Linux images before you leave!

[5] http://ophcrack.sourceforge.net

Week 11 Tasks

- Pick a password. Set *two* accounts to that.
- Dump encrypted passwords on both Win7 *and* Linux
- Try cracking both – 0phcrack[5] for Win
- Look at both encrypted forms. What can you tell?
- Also try extracting plaintext passwords on both.
- Restore Windows and Linux images before you leave!

[5] http://ophcrack.sourceforge.net

Week 11 Tasks

- Pick a password. Set *two* accounts to that.
- Dump encrypted passwords on both Win7 *and* Linux
- Try cracking both – 0phcrack[5] for Win
- Look at both encrypted forms. What can you tell?
- Also try extracting plaintext passwords on both.
- Restore Windows and Linux images before you leave!

[5] http://ophcrack.sourceforge.net

Week 11 Tasks

- Pick a password. Set *two* accounts to that.
- Dump encrypted passwords on both Win7 *and* Linux
- Try cracking both – 0phcrack[5] for Win
- Look at both encrypted forms. What can you tell?
- Also try extracting plaintext passwords on both.
- Restore Windows and Linux images before you leave!

[5] `http://ophcrack.sourceforge.net`

Week 11 Tasks

- Pick a password. Set *two* accounts to that.
- Dump encrypted passwords on both Win7 *and* Linux
- Try cracking both – 0phcrack[5] for Win
- Look at both encrypted forms. What can you tell?
- Also try extracting plaintext passwords on both.
- Restore Windows and Linux images before you leave!

[5] `http://ophcrack.sourceforge.net`