(Continue) Cryptography + (Back to) Software Security

Tadayoshi Kohno

Thanks to Vitaly Shmatikov, Dan Boneh, Dieter Gollmann, Dan Halperin, John Manferdelli, John Mitchell, Bennet Yee, and many others for sample slides and materials ...
Goals for Today

- Cryptography
- Software security (now that you’ve had more experience with Lab 1)
- HW2 out soon (on cryptography)
Note: Optimizing Exponentiation

- How to compute \( M^x \mod N \)? Say \( x=13 \)
- Sums of power of 2, \( x = 8+4+1 = 2^3+2^2+2^0 \)
- Can also write \( x \) in binary, e.g., \( x = 1101 \)
- Can solve by repeated squaring
  - \( y = 1; \)
  - \( y = y^2 \mod N \quad \text{// } y = M \)
  - \( y = y^2 \mod N \quad \text{// } y = M^2 * M = M^{2+1} = M^3 \)
  - \( y = y^2 \mod N \quad \text{// } y = (M^{2+1})^2 = M^{4+2} \)
  - \( y = y^2 \mod N \quad \text{// } y = (M^{4+2})^2 * M = M^{8+4+1} \)
- Does anyone see a potential issue?
# Timing attacks

Collect timings for exponentiation with a bunch of messages $M_1$, $M_2$, ... (e.g., RSA signing operations with a private exponent)

Assume (inductively) know $b_3=1$, $b_2=1$, guess $b_1=1$

<table>
<thead>
<tr>
<th>$i$</th>
<th>$b_i = 0$</th>
<th>$b_i = 1$</th>
<th>Comp</th>
<th>Meas</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>$y = y^2 \mod N$</td>
<td>$y = y^2 \cdot M_1 \mod N$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$y = y^2 \mod N$</td>
<td>$y = y^2 \cdot M_1 \mod N$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$y = y^2 \mod N$</td>
<td>$y = y^2 \cdot M_1 \mod N$</td>
<td>$X_1 \text{ secs}$</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>$y = y^2 \mod N$</td>
<td>$y = y^2 \cdot M_1 \mod N$</td>
<td>$Y_1 \text{ secs}$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$i$</th>
<th>$b_i = 0$</th>
<th>$b_i = 1$</th>
<th>Comp</th>
<th>Meas</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>$y = y^2 \mod N$</td>
<td>$y = y^2 \cdot M_2 \mod N$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$y = y^2 \mod N$</td>
<td>$y = y^2 \cdot M_2 \mod N$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$y = y^2 \mod N$</td>
<td>$y = y^2 \cdot M_2 \mod N$</td>
<td>$X_2 \text{ secs}$</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>$y = y^2 \mod N$</td>
<td>$y = y^2 \cdot M_2 \mod N$</td>
<td>$Y_2 \text{ secs}$</td>
<td></td>
</tr>
</tbody>
</table>
Timing attacks

- If $b_1 = 1$, then set of $\{ Y_j - X_j | j \in \{1,2, ..\} \}$ has distribution with “small” variance (due to time for final step, $i=0$)
  - “Guess” was correct when we computed $X_1, X_2, ...$

- If $b_1 = 0$, then set of $\{ Y_j - X_j | j \in \{1,2, ..\} \}$ has distribution with “large” variance (due to time for final step, $i=0$, and incorrect guess for $b_1$)
  - “Guess” was incorrect when we computed $X_1, X_2, ...$
  - So time computation wrong ($X_j$ computed as large, but really small, ...)

- Strategy: Force user to sign large number of messages $M_1, M_2, ...$. Record timings for signing.
- Iteratively learn bits of key by using above property.
Problem: How does Alice know that the public key she received is really Bob’s public key?
Distribution of Public Keys

- Public announcement or public directory
  - Risks: forgery and tampering
- Public-key certificate
  - Signed statement specifying the key and identity
    - \( \text{sig}_{\text{CA}}(“\text{Bob}”, \text{PK}_B) \)
- Common approach: certificate authority (CA)
  - Single agency responsible for certifying public keys
  - After generating a private/public key pair, user proves his identity and knowledge of the private key to obtain CA’s certificate for the public key (offline)
  - Every computer is pre-configured with CA’s public key
<table>
<thead>
<tr>
<th>Name</th>
<th>Kind</th>
<th>Date Modified</th>
<th>Expires</th>
<th>Keychain</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-Trust-nQual-01</td>
<td>certificate</td>
<td>--</td>
<td>Nov 30, 2014 3:00:00 PM</td>
<td>System Roots</td>
</tr>
<tr>
<td>A-Trust-nQual-03</td>
<td>certificate</td>
<td>--</td>
<td>Aug 17, 2015 3:00:00 PM</td>
<td>System Roots</td>
</tr>
<tr>
<td>A-Trust-Qual-01</td>
<td>certificate</td>
<td>--</td>
<td>Nov 30, 2014 3:00:00 PM</td>
<td>System Roots</td>
</tr>
<tr>
<td>A-Trust-Qual-02</td>
<td>certificate</td>
<td>--</td>
<td>Dec 2, 2014 3:00:00 PM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AAA Certificate Services</td>
<td>certificate</td>
<td>--</td>
<td>Dec 31, 2028 3:59:59 PM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AC Raiz Certicámara S.A.</td>
<td>certificate</td>
<td>--</td>
<td>Apr 2, 2030 2:42:02 PM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AddTrust Class 1 CA Root</td>
<td>certificate</td>
<td>--</td>
<td>May 30, 2020 3:38:31 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AddTrust External CA Root</td>
<td>certificate</td>
<td>--</td>
<td>May 30, 2020 3:48:38 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AddTrust Public CA Root</td>
<td>certificate</td>
<td>--</td>
<td>May 30, 2020 3:41:50 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AddTrust Qualified CA Root</td>
<td>certificate</td>
<td>--</td>
<td>May 30, 2020 3:44:50 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Admin-Root-CA</td>
<td>certificate</td>
<td>--</td>
<td>Nov 9, 2021 11:51:07 PM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AdminCA-CD-T01</td>
<td>certificate</td>
<td>--</td>
<td>Jan 25, 2016 4:36:19 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AffirmTrust Commercial</td>
<td>certificate</td>
<td>--</td>
<td>Dec 31, 2030 6:06:06 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AffirmTrust Networking</td>
<td>certificate</td>
<td>--</td>
<td>Dec 31, 2030 6:08:24 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AffirmTrust Premium</td>
<td>certificate</td>
<td>--</td>
<td>Dec 31, 2040 6:10:36 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AffirmTrust Premium ECC</td>
<td>certificate</td>
<td>--</td>
<td>Dec 31, 2040 6:20:24 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>America Onli...nation Authority 1</td>
<td>certificate</td>
<td>--</td>
<td>Nov 19, 2037 12:43:00 PM</td>
<td>System Roots</td>
</tr>
<tr>
<td>America Onli...nation Authority 2</td>
<td>certificate</td>
<td>--</td>
<td>Sep 29, 2037 7:08:00 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AOL Time W...cation Authority 1</td>
<td>certificate</td>
<td>--</td>
<td>Nov 20, 2037 7:03:00 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AOL Time W...cation Authority 2</td>
<td>certificate</td>
<td>--</td>
<td>Sep 28, 2037 4:43:00 PM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Apple Root CA</td>
<td>certificate</td>
<td>--</td>
<td>Feb 9, 2035 1:40:36 PM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Apple Root Certificate Authority</td>
<td>certificate</td>
<td>--</td>
<td>Feb 9, 2035 4:18:14 PM</td>
<td>System Roots</td>
</tr>
</tbody>
</table>
Hierarchical Approach

• Single CA certifying every public key is impractical
• Instead, use a trusted root authority
  • For example, Verisign
  • Everybody must know the public key for verifying root authority’s signatures
• Root authority signs certificates for lower-level authorities, lower-level authorities sign certificates for individual networks, and so on
  • Instead of a single certificate, use a certificate chain
    – sig_{Verisign}(“AnotherCA”, PK_{AnotherCA}), sig_{AnotherCA}(“Alice”, PK_A)
  • What happens if root authority is ever compromised?
Many Challenges

Spoofing URLs With Unicode

Posted by timothy on Mon May 27, '02 09:48 PM
from the there-is-a-problem-with-this-certificate dept.

Embedded Geek writes:

"Scientific American has an interesting article about how a pair of students at the Technion-Israel Institute of Technology registered "microsoft.com" with Verisign, using the Russian Cyrillic letters "c" and "o". Even though it is a completely different domain, the two display identically (the article uses the term "homograph"). The work was done for a paper in the Communications of the ACM (the paper itself is not online). The article characterizes attacks using this spoof as "scary, if not entirely probable," assuming that a hacker would have to first take over a page at another site. I disagree: sending out a mail message with the URL waiting to be clicked ("Bill Gates will send you ten dollars!") is just one alternate technique. While security problems with Unicode have been noted here before, this might be a new twist."
Many Challenges

http://it.slashdot.org/story/08/12/30/1655234/CCC-Create-a-Rogue-CA-Certificate
http://www.win.tue.nl/hashclash/rogue-ca/

---

**CCC Create a Rogue CA Certificate**

*Posted by [CmdrTaco](http://it.slashdot.org/users/58772) on Tue Dec 30, 2008 12:14 PM from the [they-even-faked-this-dept](http://www.win.tue.nl/hashclash/) dept.*

---

*t3rmin4t0r* writes

"Just when you were breathing easy about [Kaminsky](http://www.ciphergoth.com/), DNS and the word hijacking, by repeating the word SSL in your head, the hackers at [CCC](http://www.ccc.de) were busy at work making a hash of SSL certificate security. Here's the scoop on how they set up their own rogue CA, by (from what I can figure) reversing the hash and engineering a collision up in MD5 space. Until now, MD5 collisions have been ignored because nobody would put in that much effort to create a useful dummy file, but a CA certificate for phishing seems juicy enough to be fodder for the botnets now."
DigiNotar Hacked by Black.Spoof and Iranian Hackers

DigiNotar is a Dutch Certificate Authority. They sell SSL certificates.

Somehow, somebody managed to get a rogue SSL certificate from them on July 10th, 2011. This certificate was issued for domain name .google.com.

What can you do with such a certificate? Well, you can impersonate Google — assuming you can first reroute Internet traffic for google.com to you. This is something that can be done by a government or by a rogue ISP. Such a reroute would only affect users within that country or under that ISP.
Alternative: “Web of Trust”

- Used in PGP (Pretty Good Privacy)
- Instead of a single root certificate authority, each person has a set of keys they “trust”
  - If public-key certificate is signed by one of the “trusted” keys, the public key contained in it will be deemed valid
- Trust can be transitive
  - Can use certified keys for further certification

Alice

Friend of Alice

Friend of friend

Bob

\[
sig_{\text{Alice}}(\text{“Friend”, Friend’s key})
\]

\[
sig_{\text{Friend}}(\text{“Foaf”, FoaF’s key})
\]

I trust Alice
X.509 Certificate

- Added in X.509 versions 2 and 3 to address usability and security problems
Certificate Revocation

- Revocation is **very** important
- Many valid reasons to revoke a certificate
  - Private key corresponding to the certified public key has been compromised
  - User stopped paying his certification fee to this CA and CA no longer wishes to certify him
  - CA’s private key has been compromised!
- Expiration is a form of revocation, too
  - Many deployed systems don’t bother with revocation
  - Re-issuance of certificates is a big revenue source for certificate authorities
Certificate Revocation Mechanisms

◆ Online revocation service
  • When a certificate is presented, recipient goes to a special online service to verify whether it is still valid
    – Like a merchant dialing up the credit card processor

◆ Certificate revocation list (CRL)
  • CA periodically issues a signed list of revoked certificates
    – Credit card companies used to issue thick books of canceled credit card numbers
  • Can issue a “delta CRL” containing only updates
X.509 Certificate Revocation List

Because certificate serial numbers must be unique within each CA, this is enough to identify the certificate.
Convergence

Background observation:
• MITM attacker will have a hard time mounting man-in-the-middle attacks against all clients around the world

Basic idea:
• Lots of nodes around the world obtaining SSL/TLS certificates from servers
• Check responses across servers, and also observe unexpected changes from existing certificates
SSL
What is SSL / TLS?

- **Transport Layer Security (TLS) protocol, version 1.2**
  - De facto standard for Internet security
  - "The primary goal of the TLS protocol is to provide privacy and data integrity between two communicating applications"
  - In practice, used to protect information transmitted between browsers and Web servers (and mail readers and ...)
- **Based on Secure Sockets Layers (SSL) protocol, version 3.0**
  - Same protocol design, different algorithms
- Deployed in nearly every Web browser
SSL / TLS in the Real World

Refraction

Refraction is a free, online puzzle game for teaching fractions, developed by UW CSE's Center for Game Science.
History of the Protocol

- **SSL 1.0**
  - Internal Netscape design, early 1994?
  - Lost in the mists of time

- **SSL 2.0**
  - Published by Netscape, November 1994
  - Several weaknesses

- **SSL 3.0**
  - Designed by Netscape and Paul Kocher, November 1996

- **TLS 1.0**
  - Internet standard based on SSL 3.0, January 1999
  - Not interoperable with SSL 3.0
    - TLS uses HMAC instead of earlier MAC; can run on any port

- **TLS 1.2**
  - Remove dependencies to MD5 and SHA1
“Request for Comments”

- Network protocols are usually disseminated in the form of an RFC
- TLS version 1.0 is described in RFC 5246
- Intended to be a self-contained definition of the protocol
  - Describes the protocol in sufficient detail for readers who will be implementing it and those who will be doing protocol analysis
  - Mixture of informal prose and pseudo-code
TLS Basics

- TLS consists of two protocols
  - Familiar pattern for key exchange protocols
- Handshake protocol
  - Use public-key cryptography to establish a shared secret key between the client and the server
- Record protocol
  - Use the secret key established in the handshake protocol to protect communication between the client and the server
- We will focus on the handshake protocol
TLS Handshake Protocol

- Two parties: client and server
- Negotiate version of the protocol and the set of cryptographic algorithms to be used
  - Interoperability between different implementations of the protocol
- Authenticate client and server (optional)
  - Use digital certificates to learn each other’s public keys and verify each other’s identity
- Use public keys to establish a shared secret
Handshake Protocol Structure

- ClientHello
- ServerHello, [Certificate], [ServerKeyExchange], [CertificateRequest], ServerHelloDone
- [Certificate], ClientKeyExchange, [CertificateVerify]
- switch to negotiated cipher
- Finished

Record of all sent and received handshake messages

S

C

switch to negotiated cipher

Finished
ClientHello

Client announces (in plaintext):
- Protocol version
- Supported Cryptographic algorithms
struct {
    ProtocolVersion client_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suites;
    CompressionMethod compression_methods;
} ClientHello
ServerHello

Server responds (in plaintext) with:

- Highest protocol version supported by both client and server
- Strongest cryptographic suite selected from those offered by the client
Server sends public-key certificate containing either RSA, or Diffie-Hellman public key (depending on chosen crypto suite)
Client generates some secret key material and sends it to the server encrypted with the server’s public key (if using RSA).
"Core" SSL 3.0 Handshake (Not TLS)

If the protocol is correct, C and S share some secret key material (secret$_c$) at this point:

- Switch to key derived from secret$_c$, $N_c$, $N_s$ for C.
- Switch to key derived from secret$_c$, $N_c$, $N_s$ for S.
**Version Rollback Attack**

C, Version = 2.0, suite_c, N_c

Server is fooled into thinking it is communicating with a client who supports only SSL 2.0

Version_s = 2.0, suite_s, N_s, sig_{ca}(S,K_s), "ServerHelloDone"

C and S end up communicating using SSL 2.0 (weaker earlier version of the protocol without finished message from client)

{Secret_c}_K_s
SSL 2.0 Weaknesses (Fixed in 3.0)

- Cipher suite preferences are not authenticated
  - “Cipher suite rollback” attack is possible
- SSL 2.0 uses padding when computing MAC in block cipher modes, but padding length field is not authenticated
  - Attacker can delete bytes from the end of messages
- MAC hash uses only 40 bits in export mode
- No support for certificate chains or non-RSA algorithms, no handshake while session is open
Protocol Rollback Attacks

- Why do people release new versions of security protocols? Because the old version got broken!
- New version must be \textit{backward-compatible}
  - Not everybody upgrades right away
- Attacker can fool someone into using the old, broken version and exploit known vulnerability
  - Similar: fool victim into using weak crypto algorithms
- Defense is hard: must authenticate version in early designs
- Many protocols had “version rollback” attacks
  - SSL, SSH, GSM (cell phones)
Version Check in SSL 3.0 (Approximate)

If the protocol is correct, C and S share some secret key material secret_c at this point.

- C, Version_c=2.0, suite_c, N_c
- Version_s=2.0, suite_s, N_s
- sig_{ca}(S,K_s), “ServerHelloDone”

If “embedded” version information includes eight 3s but server supports version 3, issue error.

- "Embed" eight 3s into left side of this secret if server said Version_s=2.0
- \{Version_c, Secret_c\}_K_s

switch to key derived from secret_c, N_c, N_s

switch to key derived from secret_c, N_c, N_s
SSL/TLS Record Protection

Application Data

Fragment

Compress

Add MAC

Encrypt

Append SSL Record Header

Use symmetric keys established in handshake protocol
Summary

- **Symmetric Crypto**
  - Encryption
  - MACs
  - Dedicated Authenticated Encryption Schemes
    - GCM (Galois Counter Mode)
    - CCM
    - OCB

- **Asymmetric Crypto**
  - DH
  - RSA (encryption and signatures)
  - Authenticity of public keys

- **Protocol rollback attacks**
Symmetric Crypto
- Encryption
- MACs
- Dedicated Authenticated Encryption Schemes
  - GCM (Galois Counter Mode)
  - CCM
  - OCB

Asymmetric Crypto
- DH
- RSA (encryption and signatures)
- Authenticity of public keys

Protocol rollback attacks
Back to Software Security
Defenses

- Already discussed Stack Guard: put canary on stack
PointGuard

- **Attack**: overflow a function pointer so that it points to attack code

- **Idea**: encrypt all pointers while in memory
  - Generate a random key when program is executed
  - Each pointer is XORed with this key when loaded from memory to registers or stored back into memory
    - Pointers cannot be overflowed while in registers

- **Attacker cannot predict the target program’s key**
  - Even if pointer is overwritten, after XORing with key it will dereference to a “random” memory address
Normal Pointer Dereference [Cowan]

1. Fetch pointer value
2. Access data referenced by pointer

1. Fetch pointer value
2. Access \textit{attack code} referenced by \textit{corrupted pointer}
PointGuard Dereference

1. Fetch pointer value
2. Access data referenced by pointer

Memory

Encrypted pointer 0x7239

Decrypt

Data

Corrupted pointer 0x7239 0x1340

Decrypt

2. Access random address; segmentation fault and crash

CPU

0x1234

0x1234

0x1340

0x9786

0x9786

[ Cowan ]
PointGuard Issues

- Must be very fast
  - Pointer dereferences are very common

- Compiler issues
  - Must encrypt and decrypt only pointers
  - If compiler “spills” registers, unencrypted pointer values end up in memory and can be overwritten there

- Attacker should not be able to modify the key
  - Store key in its own non-writable memory page

- PG’d code doesn’t mix well with normal code
  - What if PG’d code needs to pass a pointer to OS kernel?
Other solutions to some of these issues

- Use safe programming languages, e.g., Java
  - What about legacy C code?
  - (Note that Java is not the complete solution)
- Program analysis of source code to find overflows
  - Coverity
  - Fortify
- Randomize stack location or encrypt return address on stack by XORing with random string
  - Attacker won’t know what address to use in his or her string
Fuzz Testing

- Generate “random” inputs to program
  - Sometimes conforming to input structures (file formats, etc)
- See if program crashes
  - If crashes, found a bug
  - Bug may be exploitable
- Surprisingly effective

- Now standard part of development lifecycle
Next slides special thanks to Hovav Shacham and Vitaly Shmatikov
Buffer Overflow: Causes and Cures

Typical memory exploit involves code injection
- Put malicious code in a predictable location in memory, usually masquerading as data
- Trick vulnerable program into passing control to it
  - Overwrite saved EIP, function callback pointer, etc.

Defense: prevent execution of untrusted code
- Make stack and other data areas non-executable
  - Note: messes up useful functionality (e.g., ActionScript)
- Digitally sign all code
- Ensure that all control transfers are into a trusted, approved code image
Mark all writeable memory locations as non-executable
- Example: Microsoft’s DEP - Data Execution Prevention
- This blocks many (not all) code injection exploits

Hardware support
- AMD “NX” bit, Intel “XD” bit (in post-2004 CPUs)
- OS can make a memory page non-executable

Widely deployed
- Windows (since XP SP2), Linux (via PaX patches), OpenBSD, OS X (since 10.5)
What Does W⊕X Not Prevent?

- Can still corrupt stack ...
  - ... or function pointers or critical data on the heap
- As long as “saved EIP” points into existing code, W⊕X protection will not block control transfer
- This is the basis of return-to-libc exploits
  - Overwrite saved EIP with address of any library routine, arrange memory to look like arguments
- May not look like a huge threat
  - Attacker cannot execute arbitrary code
  - ... especially if system() is not available
Overwritten saved EIP need not point to the beginning of a library routine

Any existing instruction in the code image is fine
• Will execute the sequence starting from this instruction

What if instruction sequence contains RET?
• Execution will be transferred... to where?
• Read the word pointed to by stack pointer (ESP)
  – Guess what? Its value is under attacker’s control! (why?)
• Use it as the new value for EIP
  – Now control is transferred to an address of attacker’s choice!
• Increment ESP to point to the next word on the stack
Chaining RETs for Fun and Profit

- Can chain together sequences ending in RET
  - Krahmer, “x86-64 buffer overflow exploits and the borrowed code chunks exploitation technique” (2005)
- What is this good for?
- Answer [Shacham et al.]: everything
  - Turing-complete language
  - Build “gadgets” for load-store, arithmetic, logic, control flow, system calls
  - Attack can perform arbitrary computation using no injected code at all!
Instruction pointer (EIP) determines which instruction to fetch and execute.

Once processor has executed the instruction, it automatically increments EIP to next instruction.

Control flow by changing value of EIP.
Return-Oriented Programming

- **Stack pointer** (ESP) determines which instruction sequence to fetch and execute
- **Processor doesn’t** automatically increment ESP
  - But the RET at end of each instruction sequence does
No-ops

- No-op instruction does nothing but advance EIP
- Return-oriented equivalent
  - Point to return instruction
  - Advances ESP
- Useful -- like a NOP sled
Immediate Constants

- Instructions can encode constants
- Return-oriented equivalent
  - Store on the stack
  - Pop into register to use
Control Flow

- Ordinary programming
  - (Conditionally) set EIP to new value
- Return-oriented equivalent
  - (Conditionally) set ESP to new value