Security 101
April 19, 2018 | Overview

Stacy Janes
Security Team, GENIVI Alliance
Software Security 101
Crypto 101
Integrity and Confidentiality

Integrity: Proving the validity of data.
Confidentiality: Protecting the contents of data.

Digital Signature
Encryption
Hashing

Unlike encryption, hashing is a “one way” function.

A hash is used to check the validity of data. It does not protect data.

Passwords should be hashed, not encrypted when stored.
Encryption – Symmetric Key

Encryption and decryption done with the same key

Symmetric cryptography is fast (relative to Asymmetric)

Key management becomes cumbersome beyond a few actors.
Encryption – Asymmetric Key

- Encryption with Public Key
- Decryption with Private Key

Asymmetric cryptography is slow (relative to Symmetric)

Private Keys are not shared

Public Keys can be shared with many actors. PKI enables this.
Digital Signature

- Encrypt with Private Key (Sign)
- Decrypt with Public Key (Verify)

X.509 Certificate around Public Key for identity verification

Does not hide data

Encrypted Hash

plaintext

digest

digital signature

Encrypt with $K_{\text{private}}$ (Sign)

Decrypt with $K_{\text{public}}$ (Verify)

X.509 Certificate around Public Key for identity verification

Does not hide data
Binary Hacking 101
Privilege Escalation

Privilege escalation is the act of exploiting a bug, design flaw or configuration oversight in the OS or an application to gain elevated access to resources that are normally protected from the application or userid.

Kernel Exploitation: Exploiting vulnerabilities in the kernel in order to gain arbitrary code execution as root. Eg: DirtyCOW

Service Exploitation: Exploiting Linux services and configuration mistakes. Eg: wildcard injection.
Vulnerabilities

Kernel panic! What are Meltdown and Spectre, the bugs affecting nearly every computer and device?

Billions of devices imperiled by new clickless Bluetooth attack

BlueBorne exploit works against unpatched devices running Android, Linux, or Windows.
“Defeating” Crypto – Easier to Bypass

Brute force is typically not a realistic attack

End point access opens up attack vectors

- Key lifting. Easy for software key if not properly protected
- Binary modification to “jam” logic branch for signature check
- Lifting clear data from memory after decryption
- Inserting malicious data to be signed/encrypted
- Shimming interfaces
“Lifting” Clear Data

Find the Decrypt function

Lift the clear data after decryption
Branch “Jamming”

Let software verify signature

Find branch that checks return code

Reverse comparison opcode to allow invalid signature to pass
"Shimming"

When an application uses a shared object, an attacker can interfere with the boundary.

Attacker uses export table of .so to generate a ‘shim’ to go between application and .so.

All data (parameters and return codes) can be siphoned and modified.
“Shimming”

When an application uses a shared object, an attacker can interfere with the boundary.

Attacker uses export table of .so to generate a ‘shim’ to go between application and .so.

All data (parameters and return codes) can be siphoned and modified.
Coding Practices
Inherited process context that is not validated like other inputs can introduce vulnerability.

Validate Inherited Process Context

Use `strncpy_s()` and `strncat_s()`

The `strncpy_s()` and `strncat_s()` functions are defined in ISO/IEC TR 24731 as drop-in replacements for `strncpy()` and `strncat()`.
Code Entanglement

• Avoid assertion checks on sensitive decisions such as a digital signature or password validation.

• “Entangle” the input value by using it to get to the asset. Eg: password is decryption key to decrypt file.

```java
pwHash = getPasswordHash();
if( pwHash == storedHash ){
    decryptFile(fn);
}
```

### Assertion Check

```java
pwHash = getPasswordHash();
if( pwHash == storedHash ){
    decryptFile(fn);
}
```

### Entangled

```java
pwHash = getPasswordHash();
decryptFile(fn, pwHash);
```
Data Parsing is Critical
<table>
<thead>
<tr>
<th>Title</th>
<th>Updated date</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strengthening Ties Between Process and Security</td>
<td>2013-07-31</td>
<td>Carol Woody</td>
</tr>
<tr>
<td>Secure Software Development Life Cycle Processes</td>
<td>2013-07-31</td>
<td>Noopur Davis</td>
</tr>
<tr>
<td>Correctness by Construction</td>
<td>2013-05-14</td>
<td>Peter Arney</td>
</tr>
<tr>
<td>Design Principles</td>
<td>2013-05-13</td>
<td>Michael Gegick, Sean Barnum</td>
</tr>
<tr>
<td>Separation of Privilege</td>
<td>2013-05-10</td>
<td>Michael Gegick, Sean Barnum</td>
</tr>
<tr>
<td>Securing the Weakest Link</td>
<td>2013-05-10</td>
<td>Michael Gegick, Sean Barnum</td>
</tr>
<tr>
<td>Reluctance to Trust</td>
<td>2013-05-10</td>
<td>Michael Gegick, Sean Barnum</td>
</tr>
<tr>
<td>Psychological Acceptability</td>
<td>2013-05-10</td>
<td>Michael Gegick, Sean Barnum</td>
</tr>
<tr>
<td>Promoting Privacy</td>
<td>2013-05-10</td>
<td>Michael Gegick, Sean Barnum</td>
</tr>
<tr>
<td>Never Assuming That Your Secrets Are Safe</td>
<td>2013-05-10</td>
<td>Michael Gegick, Sean Barnum</td>
</tr>
<tr>
<td>Least Privilege</td>
<td>2013-05-10</td>
<td>Michael Gegick, Sean Barnum</td>
</tr>
<tr>
<td>Least Common Mechanism</td>
<td>2013-05-10</td>
<td>Michael Gegick, Sean Barnum</td>
</tr>
<tr>
<td>Failing Securely</td>
<td>2013-05-10</td>
<td>Michael Gegick, Sean Barnum</td>
</tr>
<tr>
<td>Economy of Mechanism</td>
<td>2013-05-10</td>
<td>Michael Gegick, Sean Barnum</td>
</tr>
<tr>
<td>Complete Mediation</td>
<td>2013-05-10</td>
<td>C.C. Michael, Michael Gegick, Sean Barnum</td>
</tr>
<tr>
<td>Defense in Depth</td>
<td>2005-09-13</td>
<td>Sean Barnum, Michael Gegick, C.C. Michael</td>
</tr>
</tbody>
</table>
Software Protections
If software is running on a potentially hostile environment, an attacker can have full control over software execution.

Attacker can use analysis tools to detect and circumvent in-software checks.

Verification of software integrity should be done:

• At install-time
• At start-time
• During run-time
Software Protections – Transformations

- Similar to integrity checks, code transformation is useful when software is in a hostile environment.
- Code transformation can strongly mitigate static analysis of code.
- Data transformation can hide data after decryption to mitigate against siphoning.

Some form of code and data transformation is widely and expertly used by authors of sophisticated malware.

Transformation of open source can be tricky. License issues. Leakage of information through system calls.
Transforming Control Flow
System Features
ASLR

Address Space Layout Randomization

ASLR randomly arranges the **address space** positions of key data areas of a **process**, including the base of the **executable** and the positions of the **stack**, **heap** and **libraries**.

- Wikipedia

ASLR is a first line of defence against return-to-libc and ROP attacks by making it harder for attackers to know memory offsets before hand in an attack. It’s effectiveness is based on the entropy used.
Access Control

Identity

Authentication

Authorization

Unique Identifier

Passwords, biometrics

Access Control
Discretionary Access Control

• Owner of resource controls access.
• Access is based on identity or groups

• Primary attacker objective is to escalate privilege from restricted to non-restricted (root)
Mandatory Access Control

- Policy based authorization rules
- Centrally controlled by an administrator
- Users (even root) cannot override the policy
- Implemented using the LSM API
Mandatory Access Control

SELinux
- Based on labels
- Historically difficult to use

SMACK
- Designed with simplicity in mind

TOMOYO
- Uses pathnames instead of labels

AppArmor
- Uses pathnames, similar to TOMOYO
SECURITY NEEDS TO BE DESIGNED IN FROM THE BEGINNING.
Thank you!

Visit GENIVI at http://www.genivi.org or http://projects.genivi.org

Contact us: help@genivi.org