IMAGINE THE POSSIBILITIES...
SECURITY & PRIVACY

Cloud -based Dynamic Executable Verification

Rafie Shamsaasef
Director of Software Engineering
CommScope
Cloud-based Dynamic Executable Verification

- **Why**
  - Building the case for the need of such protection
- **What**
  - Defining the protection technology
- **How**
  - Diving into details
- **Where**
  - Explaining DEV applicability and use cases
- **Who**
  - Benefiting from it
- **Wrap up and Q/A**
Why code signing

Building the case for the need of such protection

- **Software Based Secure Boot**
  - Requires custom firmware (i.e. STB)
  - Can’t isolate all executable codes and 3rd party dependencies
  - Not available or suited for general software application authenticity verification
  - Can’t cover layers within the software applications
  - Actions upon failures are limited (typically not loading the app)

- **Static and Dynamic Analyses**
  - Helps in identifying and stopping tampering attacks
  - Static: analyses of the code/binary without execution
  - Dynamic: analyses of the binary in runtime
  - Need to be iterative process as part of the development cycle
  - Tools and techniques to achieve these
Building the case for the need of such protection

- Tampering Attacks and Threats
  - Relatively low effort attacks
  - Outsider and Insider attacks:
    - Compromise user’s privacy
      - Steal user’s credential or business secrets (Cryptojacking)
      - Denial of service
      - Software piracy
      - Dishonest insider

- Developers need tools/techniques to remedy against these attacks
  - Tamper-resistant code
  - Code signing in every level of software within the application
  - Anticipate certain attacks ahead of time
  - Be able to verify in runtime \(\rightarrow\) (DEV)
Dynamic Executable Verification (DEV) protection technology

- Code sign at a deeper layer (per functions and modules)
- Apply to a program during compilation by the Dynamic tamper protection toolchain
- Create hooks to check and verify code signatures in the runtime
- Randomize check points

Goals

1. Increase the cost and level of effort for performing tampering attacks
2. Lightweight
3. Tunable
4. Compatibility with standard code-signing
5. No frozen/finalized binaries
6. Support a diverse range of use-cases
7. Future-proofed formats
8. Broad platform compatibility
9. Broad toolchain compatibility
DEV design

<table>
<thead>
<tr>
<th>Build-time</th>
<th>Link-time</th>
<th>Activation</th>
<th>Runtime (not secure)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other toolchains</td>
<td>Binaries</td>
<td>Executable is unchanged post-linking</td>
<td>Executable</td>
</tr>
<tr>
<td>Source</td>
<td>Linker</td>
<td>Executable</td>
<td>Executable</td>
</tr>
<tr>
<td>Dynamic tamper protection tool</td>
<td>Protected binaries</td>
<td></td>
<td>Dynamic code signature (SEC)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© 2020 SCTE•ISBE, CableLabs & NCTA. All rights reserved. | scte.org • isbe.org
How - Dynamic Executable Verification

**Build time**

Randomly generated check functions

```
\[ f_1, f_2, f_3, \ldots, f_n \]

\[ \text{opaque jumtable} \]

\[ J(o_1), J(o_2), J(o_3), \ldots, J(o_n) \]

```

**Run time**

```
\[ h_1, h_2, h_3, \ldots, h_n \]

\[ \text{opaque jumtable} \]

\[ J(o_1), J(o_2), J(o_3), \ldots, J(o_n) \]

```

\[ \text{bootstrap} \]
Motivations for cloud-based DEV

- Add level of security to the process of code signing
  - Request code signing only by authorized parties
  - Verify and certify the code signing by security architect(s) or manager(s)
- Offer detailed reporting, metrics and security alerts
  - Layers access to various reports
  - Track protection over time and produce metrics for future analyses
- Prevent the source of tampering
  - No way to input manually generated "dynamic code signing certificate (CER)"
- Get assurance of the DEV-enabled software per each release
- Have an iterative process in place
  - Static/Dynamic analyses as part of holistic application protection
Cloud-based DEV design
Explaining DEV applicability and use cases

- Any software, any platform!
- In-App contained self-checking protection in runtime
- Examples
  - Browser-based App
  - Container-based App
  - DRM App
  - Many others

Benefiting from it

- Security engineers
- Software developers
- Software managers
- Release/Deployment engineers
- Customers!
DEV: Dynamic Executable Verification

- DEV is a dynamic code signing and verification technology that can be applied to any software application and protects against tampering attacks.
- DEV is assigned during the build time and verified at run time.
- DEV randomly distributes signatures throughout the binary to make it hard to trace.
- DEV has very low overhead during run time dynamic verification.
- DEV can be utilized by vast majority of software applications built with LLVM.
- Cloud-based DEV gives better control and oversight to the protection practice.
- Cloud-based DEV also creates a clear separation of software development vs protection.

N-Mesh Security Suite
Thank You!

Rafie Sham saasef
Director of Software Engineering
CommScope
rafie.shamsaasef@commscope.com

Aaron Anderson
Security Architect
CommScope
aaron.anderson@commscope.com

Sasha Medvinsky
Engineering Fellow
CommScope
sasha.medvinsky@commscope.com