Security Team

May 9, 2017 | Overview

Stacy Janes

Security Team Lead, GENIVI Alliance

Security Training

Security Education

Existing vulnerabilities — See what happens when coding errors get into the wild

How to hack a binary — Some basics of log and trace, ELF headers, binary analysis

Secure Coding Guidelines — Learn some of the concepts of writing secure products

Static Code Analysis — Learn the benefits of static analysis tools.

Security Education

How to review 3rd party software
How to mitigate the risk of 3rd party software.

Software Security technologies
What software based security technologies exist and what they are designed to do.

Hardware Security technologies
Hardware security. What does it do and why to use it.

Threat models
Want to create your own attack trees? Here's how we do it.

GENIVI®

Security Training – Day Schedule

9:30-10:00	Existing Vulnerabilities (Ben)
10:00-10:30	How to Hack a Binary (Jeremiah)
10:30-11:00	break
11:00-11:30	Secure Coding Guidelines (Assaf)
11:30-12:00	Static Code Analysis (Sergiu)
12:00-12:30	How to Review 3rd Party Software (Sergiu, Ted)

12:30-14:00	lunch
14:00-14:30	Software Security (Stacy, Assaf)
14:30-15:00	Hardware Security (Erik)
15:00-15:30	Threat Models (Ben)

GENIVI®

Existing Vulnerabilites

May 2017 | A Look at the Recent History of Vulnerabilities in Linux (and how a little typo can make everything go wrong)

Ben Gardiner

Principal Security Engineer, Irdeto

Agenda

- 20 minutes:
- Exposition of vulnerabilities in Linux (the kernel or the ecosystem) from recent history
- Non-exhaustive (we only have 20 minutes)
- Focus on ‘what went wrong?’ and ‘what was the impact?’

Heartbleed

- OpenSSL-served sockets leak data from freed memory to unauthenticated clients.
- 2012-2014

Heartbleed (impacts)

- Remote attackers can siphon nearly anything from memory that wasn't sanitized before being freed
  - e.g. private keys
  - passwords

GENIVI®

Heartbleed (fix, abridged)

```
 /* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
+ if (1 + 2 + payload + 16 > s->s3->rrec.length)
+ return 0; /* silently discard per RFC 6520 sec. 4 */
pl = p;
…
/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
```

theregister/…/heartbleed_explained/

GENIVI®

Heartbleed (conclusion)

- Assume input data is attacker-controlled / Don’t trust input data.

Shellshock

- Privilege Escalation enabling attacker to run code in the context of the shell script whose input they control
- 1989-2014

Shellshock (impacts)

- CGI webserver scripts
- DHCP Clients
- OpenSSH (ForceCommand)

- Also spurred a slew of other bash-bugs (CVEs-2014-6271 6277 6278 7169 7186 7186 7187)

GENIVI®

Shellshock (sample exploits)

```
$env x='() { :;}; echo vulnerable' bash -c "echo this is a test"
vulnerable
this is a test
$
```

```
$env X='() { (a)=>\' bash -c "echo date"; cat echo
bash: X: line 1: syntax error near unexpected token `='
bash: X: line 1: `'
bash: error importing function definition for `X'
Wed Apr  5 18:28:48 PDT 2017
$
```

Shellshock (conclusion)

- Parsing is hard / Fuzz your own parsers and/or implement the parsing code in memory safe, provably correct ways.

ImageTragick

- Parser bugs in ImageMagick can lead to Remote Code Execution (RCE) – because ImageMagick is used by lots of websites to proces user-submitted graphics

GENIVI®

ImageTragick (impacts)

- Forums Posts and Profiles
- Social Media Sites Uploads and Profiles
- Album Art on Media Players (e.g Headunits)

ImageTragick (sample exploit)

exploit.mvg:

```
push graphic-context viewbox 0 0 640 480 fill
 'url(https://example.com/image.jpg";|ls "-la)'
 pop graphic-context
```


GENIVI®

ImageTragick (sample exploit 2)

```
# hexdump -C rce1.jpg | head
00000000  70 75 73 68 20 67 72 61  70 68 69 63 2d 63 6f 6e  |push graphic-con|
00000010  74 65 78 74 0a 76 69 65  77 62 6f 78 20 30 20 30  |text.viewbox 0 0|
00000020  20 36 34 30 20 34 38 30  0a 66 69 6c 6c 20 27 75  | 640 480.fill 'u|
00000030  72 6c 28 68 74 74 70 73  3a 2f 2f 31 32 37 2e 30  |rl(https://127.0|
00000040  2e 30 2e 30 2f 6f 6f 70  73 2e 6a 70 67 22 7c 74  |.0.0/oops.jpg"|t|
00000050  6f 75 63 68 20 22 72 63  65 31 29 27 0a 70 6f 70  |ouch "rce1)'.pop|
00000060  20 67 72 61 70 68 69 63  2d 63 6f 6e 74 65 78 74  | graphic-context|
00000070  0a                                                |.|
00000071
```

```
# identify rce1.jpg
identify: unrecognized color `https://127.0.0.0/oops.jpg"|touch "rce1' @ warnir
identify: unable to open image `/var/tmp/magick-49419pGsK2PNsCdcQ': No such fil
identify: unable to open file `/var/tmp/magick-49419pGsK2PNsCdcQ': No such file
rce1.jpg MVG 640x480 640x480+0+0 16-bit sRGB 113B 0.000u 1:15.490
identify: non-conforming drawing primitive definition `fill' @ error/draw.c/Dr
```

```
# ls rce1
rce1
```


GENIVI®

ImageTragick (conclusions)

- Parsing is (still) hard / Really focus on those parsers

GENIVI®

DirtyCOW

- A race in the Copy-On-Write logic of the Kernel
- The winner gets to write to pages (they might not otherwise have write access to)
- From 2007 to 2016

GENIVI

DirtyCOW (impacts)

- Write to file nomally unmodifable by the current user
- Applied: Escalate Privileges to root
  - E.g. root (Android) phones
  - Break out of containers/sandboxes
  - Many many more

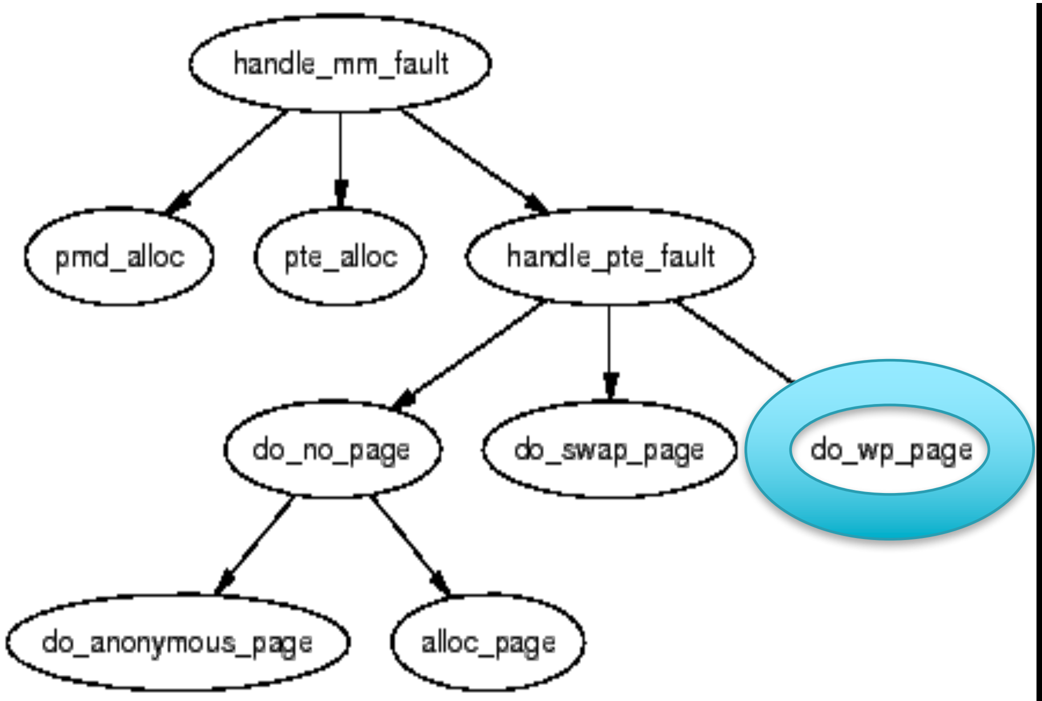
DirtyCOW (fix, summary)

“To fix it, we introduce a new internal FOLL_COW flag to **mark the "yes, we already did a COW"** rather than play racy games with FOLL_WRITE that is very fundamental, and **then use the pte dirty** flag to validate that the FOLL_COW flag is still valid.” -- Linus

kernel.git/…/19be0eaffa3ac7d8eb6784ad9bdbc7d67ed8e619

GENIVI®

DirtyCOW (exploit, summary)




```c
void *madviseThread(void *arg)
{
  int i,c=0; for(i=0;i<100000000;i++)
    c+=madvise(map,100,MADV_DONTNEED);
}

void *procselfmemThread(void *arg)
{
  int f=open("/proc/self/mem",O_RDWR);
  int i,c=0; for(i=0;i<100000000;i++) {
    lseek(f,(uintptr_t) map,SEEK_SET);
    c+=write(f,str,strlen(str));
  }
}

int main(int argc,char *argv[])
{
  map=mmap(NULL,st.st_size,PROT_READ,MAP_PRIVATE,f,0);
  pthread_create(&pth1,NULL,madviseThread,argv[1]);
  pthread_create(&pth2,NULL,procselfmemThread,argv[2]);
...
```

from PoC Exploit

DirtyCOW (conclusion)

- Concurrency is hard / Use lock checkers and/or designs that are provably correct
- But Also: Assume that the gatekeeper can be compromised / Design your defenses against root.

Conclusions

- Treat input as attacker-controlled
- A stray pointer might not crash your program -- it might give away secret info instead
- Parsers are hard.
- No, really: parsers are hard. Fuzz them.
- Concurrency can kill;
- But, more importantly: don't trust your access control gatekeepers (including the kernel).

Hacking Linux Binaries

May 2017 | Using ELF headers to make binaries do weird things

Jeremiah C. Foster

Open Source Technologist Pelagicore
Community Manager GENIVI

Agenda

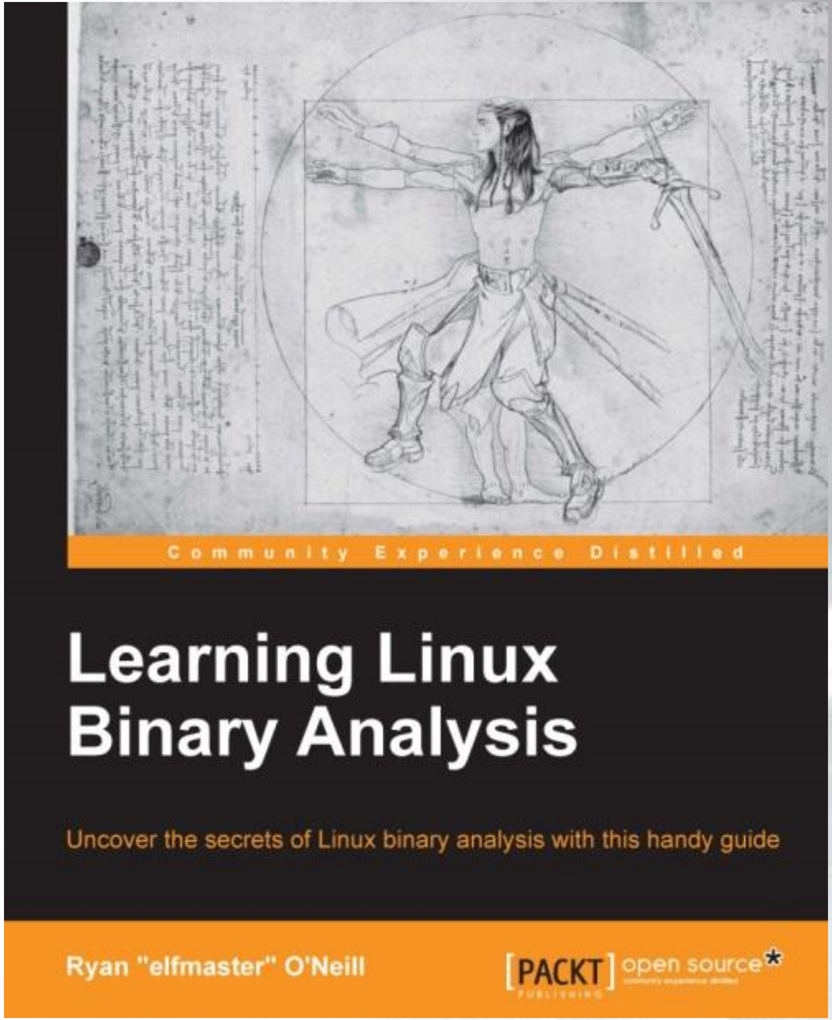
- 20-ish minutes:
- Can you crack a GNU/Linux binary for fun and profit?
  - Yes
- How?
  - Lemme show you
- This is only one way to do it, there are other ways to compromise binaries or a GNU/Linux system
- This also assumes that there is no MAC, SELinux, AppArmour, . . .

How do you ‘crack’ a linux binary?

- One might begin with ‘fingerprinting’ the binary much like one fingerprints a web site. That is to say, gather as much information as possible to determine if there are already known exploits.

- A set of tools is highly useful for this type of work
  - GDB -- GNU debugger
  - strace -- system call trace
  - ltrace -- library trace
  - objdump and objcopy -- from GNU binutils
  - readelf

GENIVI®


Community Experience Distilled
Learning Linux Binary Analysis
Uncover the secrets of Linux binary analysis with this handy guide
Ryan "elfmaster" O'Neill
[PACKT] open source*
PUBLISHING

GENIVI®

GDB GNU debugger, ptrace

- Works best with debugging symbols and, like ptrace and other tools it is an assisted application.
  - Binaries are often ‘stripped’ of their debugging symbols making them harder to reverse engineer. The ‘file’ command can tell if a binary is stripped
- ptrace is a Linux system call that can attach to a process address space and modify it.
  - This too requires a good deal of manual intervention. ptrace is in the kernel, so you’ll need to have elevated permission already to use it. Since GENIVI code is delivered as source that means that nearly anyone can do this however.

GENIVI®

Debugging example

```c
#include <stdio.h>

int main (void)
{
  printf ("Hello world.\n");

  numb();
  return 0;
}

int numb (void)
{
  printf ("hello too\n");
  return 0;
}
```

```
jefo-debian:~/code/C>  ./debugme
Hello world.
hello too

Starting program:
/home/jeremiah/code/C/debugme
Hello world.

Breakpoint 1, main () at debugme.c:7
7        numb();
```

strace -- trace system calls and signals

strace “intercepts and records the system calls which are called by a process and the signals which are received by a process. The name of each system call, its arguments and its return value are printed on standard error”

```
. . .
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\320\3\2\0\0\0\0\0"..., 832) = 832
. . .
```

GENIVI®

Disassembly

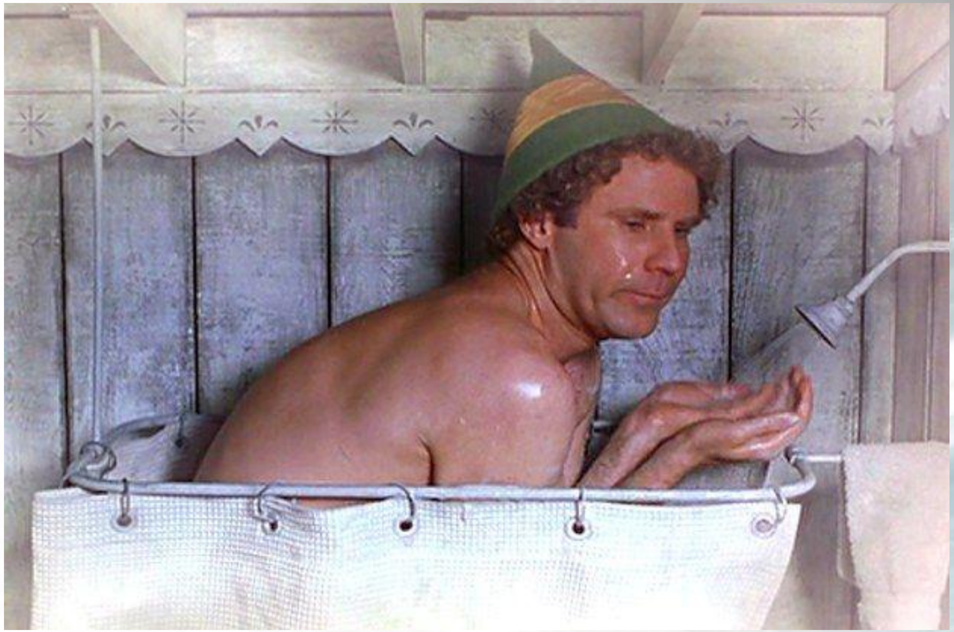
```
$ objdump -D ./debugme > debugme.asm
```

Dump out assembler code from your binary

```
$ objdump -Tt ./debugme
```

Dump out symbols

Let’s look at ELF



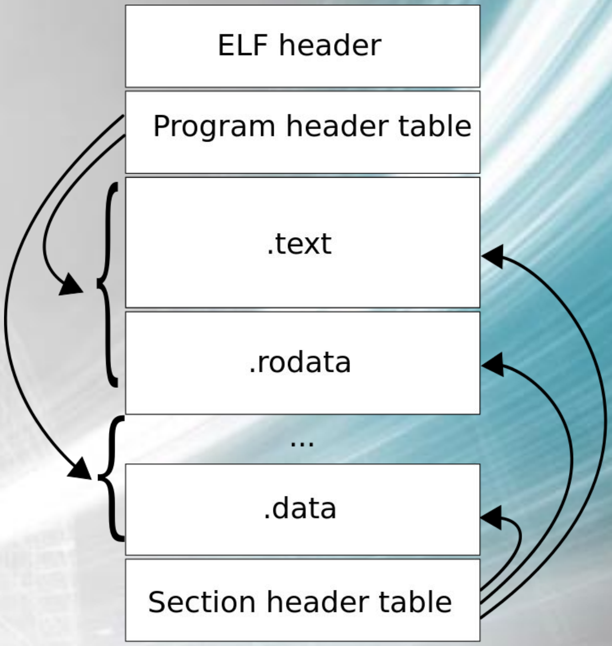
GENIVI®

Executable and **L**inkable **F**ormat

A common standard file format for executable files, object code, shared libraries, and core dumps. First published in the specification for the application binary interface (ABI) of the UNIX operating system.

Used in Linux, Solaris, QNX, FreeBSD, Playstation 3 & 4, Android since Lollipop 5.0, Windows Subsystem for Linux, and a lot more.

(wikipedia)



GENIVI®

Memory

- An ELF executable is nearly the same in memory as it is on disk with the exception of changes to the .bss section.
- The .bss section the data section that holds the length of the local variables, but not the values.
- Peter van der Linden, a C programmer and author, says, "Some people like to remember it as 'Better Save Space.' Since the BSS segment only holds variables that don't have any value yet, it doesn't actually need to store the image of these variables. The size that BSS will require at runtime is recorded in the object file, but BSS (unlike the data segment) doesn't take up any actual space in the object file."

Memory

Complex malware can live in memory and remain undetected as it is very hard to find

- Since virus and rootkit techniques used in ELF binares can also be applied to runtime code, hackers prefer to remain hidden and use various techniques;
    - GOT infection
    - Procedure linkage table infection
    - Function trampolines
    - Shared library injection
    - Relocatable code injection
    - Direct modification to the text segment

GENIVI®

Resources

- http://www.bitlackeys.org/ -- Ryan “Elfmaster” O’Neill’s own web site
- http://vxheaven.org/lib/vrn00.html -- Modern Day ELF Runtime infection via GOT poisoning

Secure Coding

May 10, 2017 | GENIVI Security Team

Assaf Harel

Co-Founder and CTO, Karamba Security

Defensive Coding –

Understanding how attackers think

Attackers approach

- Based on my experience managing Karamba's Red Team

- Attackers will always look for low hanging fruits
  - Open ports (using nmap)
  - Easy passwords

  - Boot sequence
  - JTAG / Serial ports

Secure coding mitigations

- Based on my experience managing Karamba's Red Team

- Attackers will always look for low hanging fruits
  - Open ports (using nmap) –-- **Authentication & Encryption**
  - Easy passwords –-------------- **Different & Strong passwords or other authentications**
  - Boot sequence ---------------- **Hardware based secure boot**
  - JTAG / Serial ports ------------ **Remove ports or secure the protocol when impossible**

Attackers approach

- When reviewing code
  - They will prefer closed source over open source
  - Look for memcpy() / strcpy() – buffer overflows

Secure coding mitigations

- When reviewing code
  - They will prefer closed source over open source
    - **Use well maintained open source modules**
    - **Update frequently and follow the security mailing lists**
  - Look for memcpy() or strcpy() – buffer overflows
    - **Use secure API flavors e.g. memCcpy() / strncpy()**

Attackers approach

- Black box research (reverse engineering)
  - Obfuscation is an annoying obstacle
  - ASLR, Canaries, NX, Heap protectors are an annoying obstacle
  - Tools like IDA makes your code completely readable from binary

Secure coding mitigations

- Black box research (reverse engineering)
  - Obfuscation is an annoying obstacle – **Good Practice**
  - ASLR, Canaries, NX, Heap protectors are an annoying obstacle - **Good Practice**
  - Tools like IDA makes your code completely readable from binary – **Prefer data Encryption over Obfuscation, use only public keys in the code**

Attackers approach

- ROP attacks
  - Kept short
  - Either from in-process memory or from libc memory

  - Used to obtain code execution and run reverse shell (i.e. the shell from the device connects to the attacker C&C)

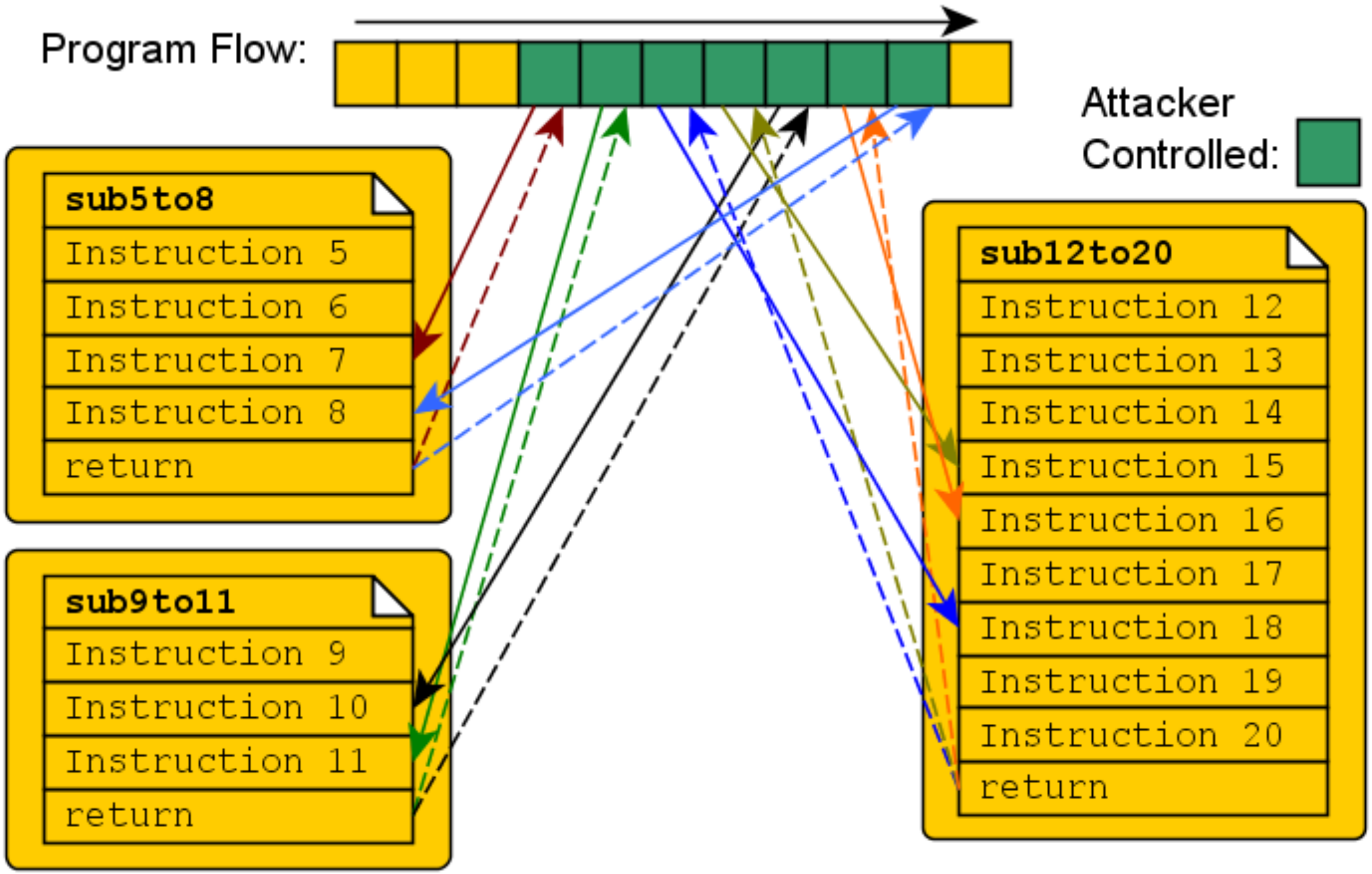
Understanding ROP

- Execute machine instruction sequences ("**gadgets**")

- **Gadgets** ends with **return**, and are located in existing libraries

- Chained together, gadgets allow performing arbitrary operations
  - In **libc** sufficient gadgets exist for **Turing-complete functionality**

Understanding ROP



Attackers approach

- ROP attacks
  - Kept short
  - Either from in-process memory or from libc memory

  - Used to obtain code execution and run reverse shell (i.e. the shell from the device connects to the attacker C&C)

Secure coding mitigations

- ROP attacks
  - Kept short
  - Either from in-process memory or from libc memory –

    CFI tools fight ROP attacks and dramatically reduce the amount of ROP gadgets
  - Used to obtain code execution and run reverse shell (i.e. the shell from the device connects to the attacker C&C)

Attackers approach

- Privilege escalation
  - Hundreds of Kernel CVEs
  - The 2nd step of the attack (1st step is code execution)
  - Access Control tools are an annoying obstacle

Secure coding mitigations

- Privilege escalation
  - Hundreds of Kernel CVEs
  - The 2nd step of the attack (1st step is code execution)
  - Access Control tools are an annoying obstacle

Good Practice

Secure Coding best practices

https://www.securecoding.cert.org/confluence/display/seccode/Top+10+Secure+Coding+Practices

Secure Coding best practices

1. Validate input
2. Use effective quality assurance techniques
3. Architect and design for security
4. Model threats
5. Default deny & the principle of least privilege
6. Practice defense in depth
7. Pay attention to compiler warnings
8. Adopt a secure coding standard

Validate input

- Be suspicious of:
  - Command line arguments
  - Network interfaces
  - Environmental variables
  - User controlled files

Use effective quality assurance techniques

- Fuzz testing
- Penetration testing
- Source code audits
- Independent security reviews
  - Bring an independent perspective identifying and correcting invalid assumptions

Architect and design for security

- Architect and design your software to enforce security policies
- Define security requirements early in the development life cycle
- Keep it simple
    - Complex designs => security mechanisms become more complex

Model threats

- Try to anticipate the threats:
  - Attacker objectives
  - Key assets
  - Threats to each asset or component
  - Rate the threats based on risk ranking
  - Develop threat mitigation strategies
    - (designs, code, and test cases)

Default deny & the principle of least privilege

- Access decisions based on permission not exclusion
  - The default is that access is denied and the scheme identifies when access is permitted
- Processes should execute with the least set of privileges necessary to complete their job
  - Elevated permission should be held for minimum time

Practice defense in depth

- Manage risk with multiple defensive strategies
  - When one layer is inadequate, another layer can prevent

Pay attention to compiler warnings

- Use the highest warning level available
- Eliminate warnings by modifying the code
- Use static and dynamic analysis tools to eliminate additional security flaws

Adopt a secure coding standard

- Develop and/or apply a secure coding standard for your target development language and platform
  - Cert
  - Open Web Application Security Project (OWASP)
  - Berkeley
  - Oracle
  - Microsoft

GENIVI®

Security Training

May 10, 2017 | Static Code Analysis

Sergiu ZAHARIA

Technology Architect BearingPoint, GENIVI Alliance

Static Code Analysis

Content

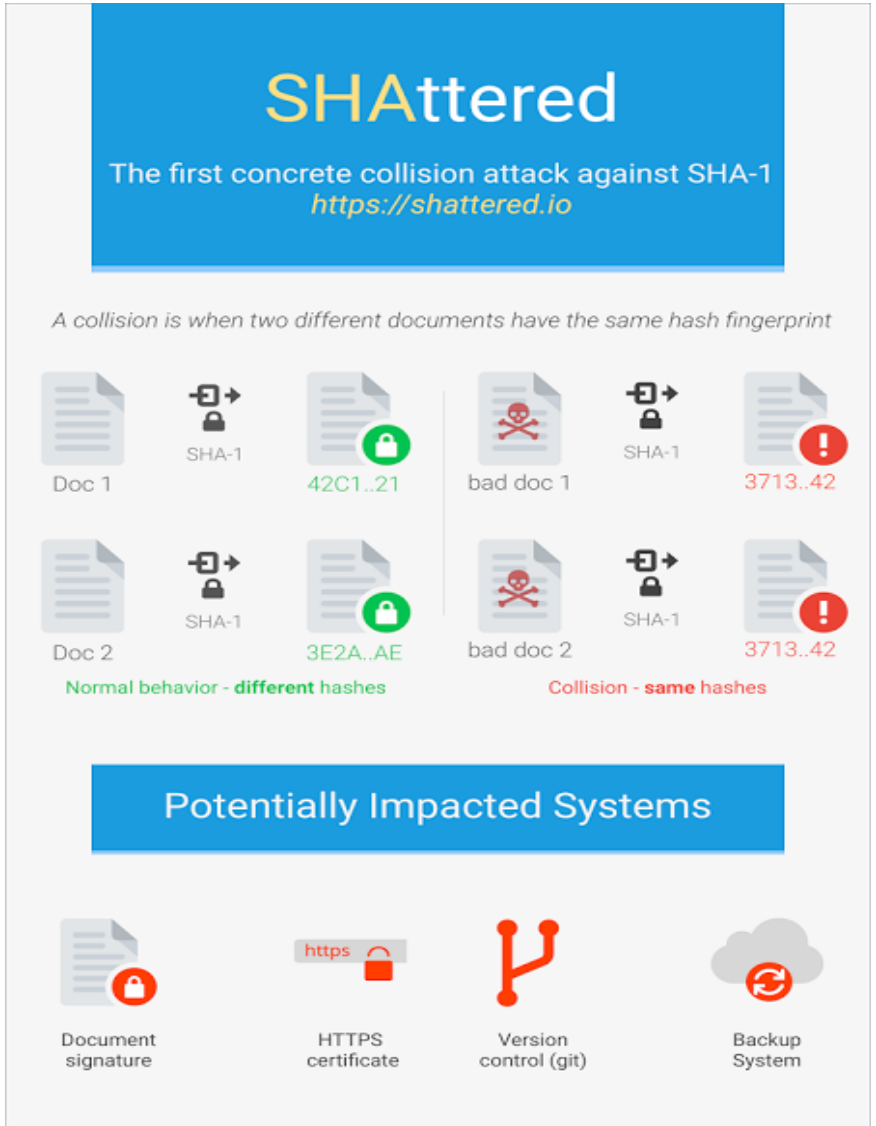
- Why security standards
- What can SAST scanners identify
- Deep dive into some vulnerabilities
- Solutions at code level and process level

Why Security Standards

February 23, 2017
Announcing the first SHA1 collision
https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html

"Today, more than 20 years after of SHA-1 was first introduced, we are announcing the first practical technique for generating a collision."

We are not all cryptologists!



Why Security Standards

Benchmark
OWASP Top 10
OWASP Top 10 Mobile
PCI DSS
Mitre CWE
SANS Top 25
FISMA
HIPAA
MISRA
BSIMM
NIST SP 800-53
DISA STIG 4.1
WASC 2.0

- Many security standards
- Mandatory or not, we have to follow them
- Groups of experts do a great job for us
- Not easy to know details of all standards
- SAST solutions use them when reviewing code

What can SAST scanners identify

- Vulnerabilities in the code (sample from Find Security Bugs):

```
MessageDigest sha1Digest = MessageDigest.getInstance("SHA1");
```

- How these vulnerabilities are propagated in the application

```
sha1Digest.update(password.getBytes());
byte[] hashValue = sha1Digest.digest();
```

- Which security standards are not fulfilled

OWASP Top 10, SANS Top 25

Types of findings (from a tool)

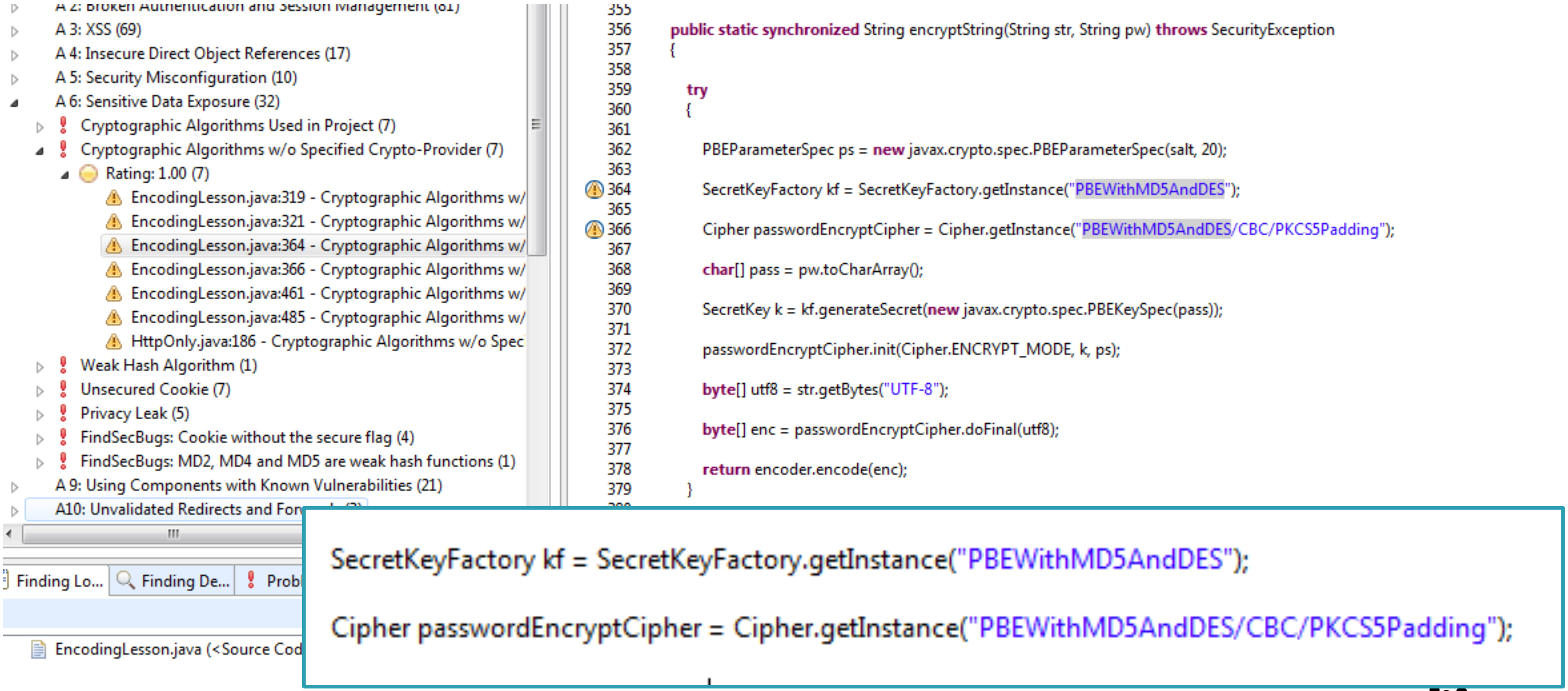
- WebGoat vulnerable application analyzed
- Findings based on Top 10 OWASP
- Most of us know about command injection, SQL injection, hardcoded passwords and buffer overflow vulnerabilities
- Let's see in detail cryptology related findings which otherwise would pass undetectable
- Cryptology is vital for automotive code; so risk ratings given by SAST solutions may be lower than real risk level

500 findings listed, 0 filtered
Analyzed: 2017-03-31 10:35:02

Tags:OWASP Top 10	Problem Type	Rating	<>	Category	Classification	CWE Number	Reviewed State	Date

- A 1: Injection (74)
- A 2: Broken Authentication and Session Management (81)
- A 3: XSS (69)
- A 4: Insecure Direct Object References (17)
- A 5: Security Misconfiguration (10)
- A 6: Sensitive Data Exposure (32)
- A 9: Using Components with Known Vulnerabilities (21)
- A10: Unvalidated Redirects and Forwards (3)
- <none> (193)
  - Applied Java Reflection (4)
  - Usage of 'java.util.Random' (2)
  - IO Stream Resource Leak (71)
  - Socket Resource Leak (2)
  - Trust Boundary Violation: HTTP Session (5)
  - FindSecBugs: Cipher is susceptible to Padding Oracle (2)
  - FindSecBugs: Cipher with no integrity (2)
  - FindSecBugs: Cookie without the HttpOnly flag (4)
  - FindSecBugs: Potential XPath Injection (1)
  - FindSecBugs: Predictable pseudorandom number generator (2)
  - FindSecBugs: Regex DOS (ReDOS) (2)
  - FindSecBugs: Tainted filename read (6)
  - Findbugs: Class defines equals() and uses Object.hashCode() (2)
  - Findbugs: Class inherits equals() and uses Object.hashCode() (73)
  - Findbugs: Field isn't final and can't be protected from malicious code (1)

Exemplification on crypto-concepts



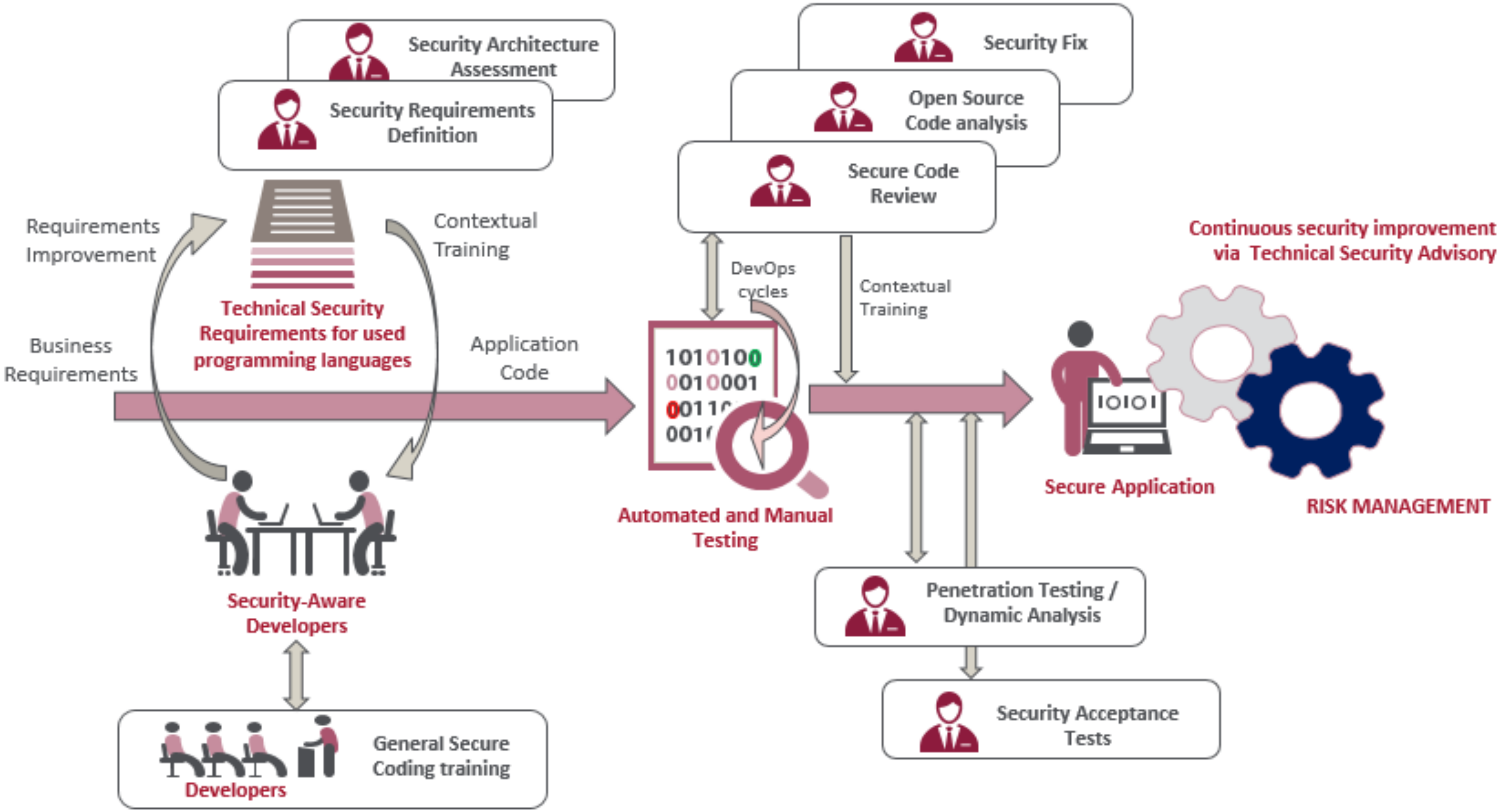
Exemplification on crypto-concepts. Solution

After several minutes of research:

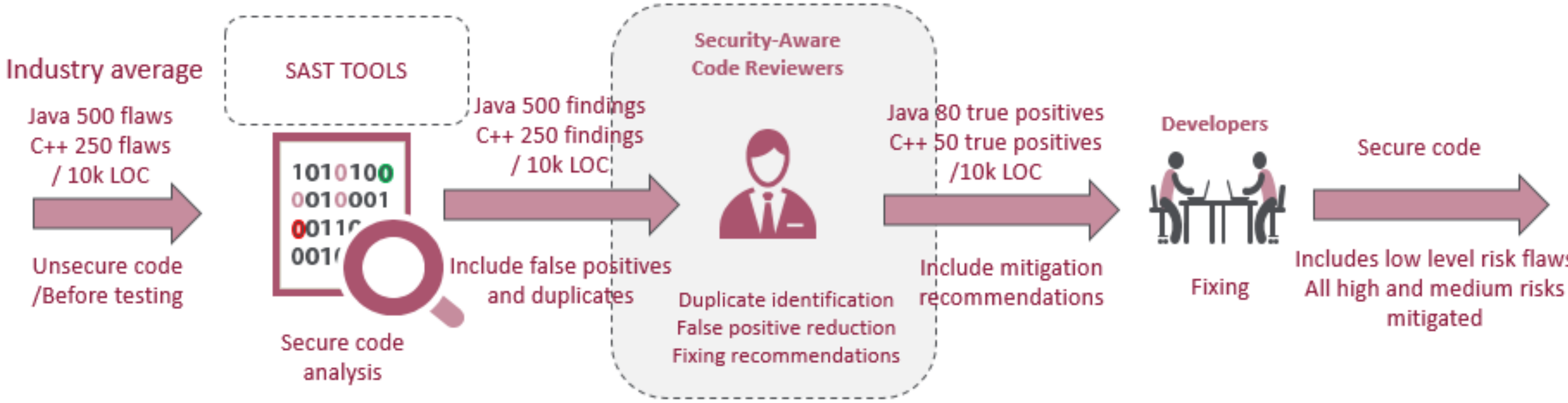
Bouncy Castle is a powerful and complete cryptography package.

```
StandardPBEStringEncryptor myFirstEncryptor = new StandardPBEStringEncryptor();
myFirstEncryptor.setProvider(new BouncyCastleProvider());
myFirstEncryptor.setAlgorithm("PBEWITHSHA256AND128BITAES-CBC-BC");
```

Holistic Solutions for Application Security



Zoom on Secure Code Review Process



False Positives Reduction	The security-aware code reviewers analyze the findings provided by SAST and identifies the ones which are not posing a real security risk (false positives) and the duplicates, based on the context provided by the application type, the IDE content and/or the involved developers. The code reviewers have access to all code required by a qualitative analysis of true positives.
True Positives Fixing Recommendation	For the true positives findings, the security-aware code reviewers provide fixing recommendations according to the programming language, context and type of flaw. The fixing process is implemented by the developers.

Which languages can SAST solutions cover?

- Most of SAST tools cover between 1-3 languages like Java, C/C++, C#
- Some of them are freeware solutions
- There are commercial solutions covering most common languages and IDE/Build integration capabilities
- Let’s see quickly how SAST tools work!

Programming Language	TOOL 1	TOOL 2
Java	Yes	Yes
.NET	Yes	Ye
C#	Yes	Ye
JavaScript	Yes	Ye
C/C++	Yes	Ye
Go (Golang)	No	N
Python	Yes	Ye
Ruby	Yes	Ye
ObjectiveC	Yes	Ye
SQL	Yes	Ye
XML	Yes	Ye
HTML5	Yes	Ye
AngularJS	No	Ye
JEE	Yes	Ye
Django	Yes	Ye
JavaServer Faces JSF	Yes	Ye
Jersey	Yes	N
Spring	Yes	Ye

Programming Language	TOOL 1	TOOL 2
Grails	No	Yes
Apigee	No	No
Scala	No	Yes
Groovy	No	Yes
Bash/Shell Scripting	No	No
TypeScript	No	No
PHP	Yes	Yes
VB	Yes	Yes
Perl	No	Yes
Xamarin	No	No
XAML	No	Yes
Universal framework	Yes	No
Player framework	No	No
Galasoft mvvm light	No	No
ABAP/BSP	Yes	No
ActionScript/MXML (Flex)	Yes	No
Clasic ASP (with VBScript)	Yes	Yes
Cobol	Yes	No
ColdFusion CFML	Yes	No
JSP	Yes	Yes
Swift	Yes	Yes

GENIVI®

Security Training

May 10, 2017 | Free and Open Source Software Security

Sergiu ZAHARIA

Technology Architect BearingPoint, GENIVI Alliance

Free and Open Source Software Security

Content

- What means FOSS / Security?
- What developers can do?
- How automatic tools help?
- The holistic approach around FOSS / Security

What means FOSS(S)ecurity and why we need it?

FOSS(S) = Scanning and indexing the entire library of freeware and open source components, to identify the already published vulnerabilities related to those components.

Components	Known Critical or Severe Security Vulnerabilities	Known restrictive licenses
106	24	9

Sonatype 2014 analysis of Application Health Check results (for an average application)

Do you remember the Bouncy Castle package?

CVE Details
The ultimate security vulnerability datasource

Bouncycastle : Security Vulnerabilities

CVSS Scores Greater Than: 0 1 2 3 4 5 6 7 8 9
Sort Results By : CVE Number Descending CVE Number Ascending CVSS Score Descending Number Of Exploits Descending
Copy Results Download Results

#	CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score	Gained Access Level	Access	Complexity	Authentication	Conf.	Integ.	Avail.
1	CVE-2016-2427	200		+Info	2016-04-17	2016-08-18	4.3	None	Remote	Medium	Not required	Partial	None	None
	** DISPUTED ** The AES-GCM specification in RFC 5084, as used in Android 5.x and 6.x, recommends 12 octets for the aes-ICVlen parameter field, which might make it easier for attackers to defeat a cryptographic protection mechanism and discover an authentication key via a crafted application, aka internal bug 26234568. NOTE: The vendor disputes the existence of this potential issue in Android, stating "This CVE was raised in error: it referred to the authentication tag size in GCM, whose default according to ASN.1 encoding (12 bytes) can lead to vulnerabilities. After careful consideration, it was decided that the insecure default value of 12 bytes was a default only for the encoding and not default anywhere else in Android, and hence no vulnerability existed."													
2	CVE-2015-7940	310			2015-11-09	2016-12-07	5.0	None	Remote	Low	Not required	Partial	None	None
	The Bouncy Castle Java library before 1.51 does not validate a point is withing the elliptic curve, which makes it easier for remote attackers to obtain private keys via a series of crafted elliptic curve Diffie Hellman (ECDH) key exchanges, aka an "invalid curve attack."													
3	CVE-2013-1624	310			2013-02-08	2014-04-19	4.0	None	Remote	High	Not required	Partial	Partial	None
	The TLS implementation in the Bouncy Castle Java library before 1.48 and C# library before 1.8 does not properly consider timing side-channel attacks on a noncompliant MAC check operation during the processing of malformed CBC padding, which allows remote attackers to conduct distinguishing attacks and plaintext-recovery attacks via statistical analysis of timing data for crafted packets, a related issue to CVE-2013-0169.													
4	CVE-2007-6721				2009-03-29	2012-11-15	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
	The Legion of the Bouncy Castle Java Cryptography API before release 1.38, as used in Crypto Provider Package before 1.36, has unknown impact and remote attack vectors related to "a Bleichenbacher vulnerability in simple RSA CMS signatures without signed attributes."													

https://www.cvedetails.com/vulnerability-list/vendor_id-7637/Bouncycastle.html

What should developers do?

NEWS

Java Release 1.56 is now available for download.

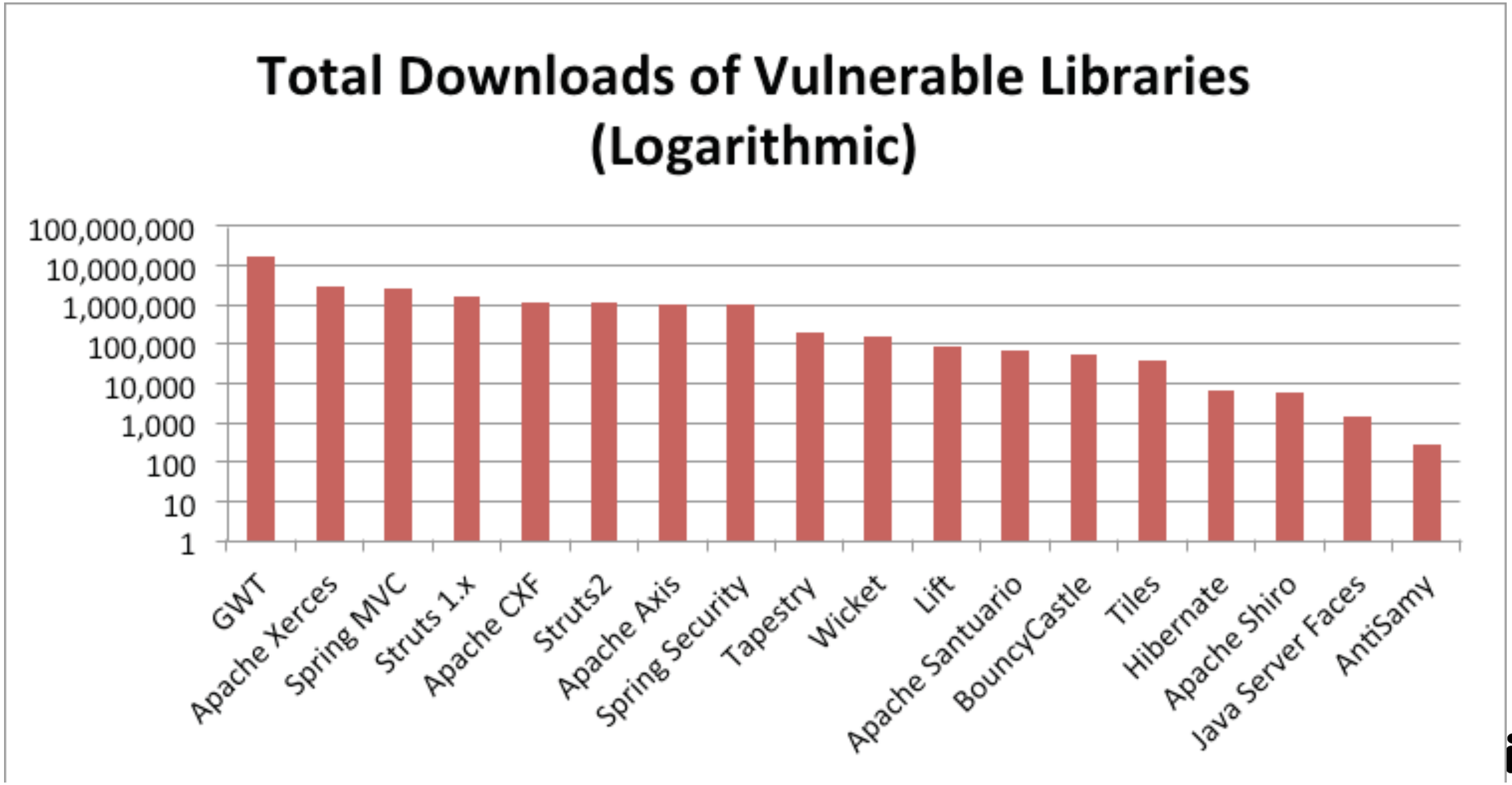
Friday 23rd December 2016

“The Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms.
The package is organized so that it contains a light-weight API suitable for use in any environment (including the J2ME) with the additional infrastructure to conform the algorithms to the JCE framework. “

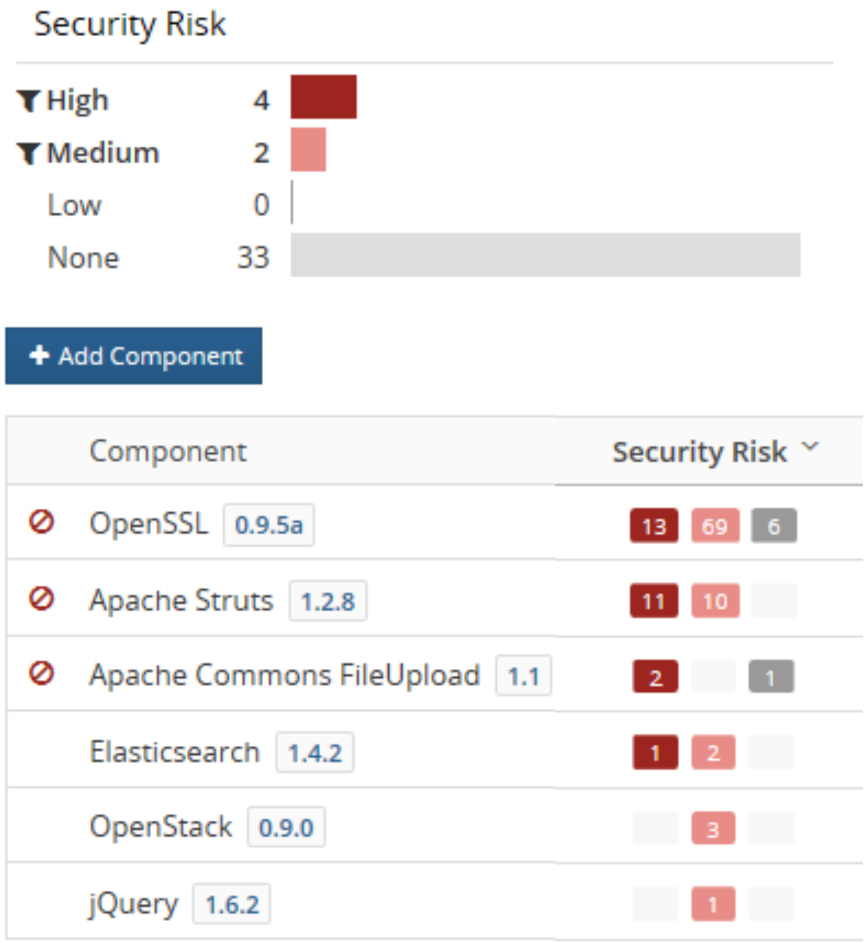
- Each FOSS component may add critical vulnerabilities to the application code
- Developers should be aware of each version of each component and their level of risk
- Or maybe not… and use some automatic tools and processes to do this work in background
- FOSS(S) tools estimate the application security and compliance risk based on used components

Are we really vulnerable?

CONTRAST SECURITY "The Unfortunate *Reality* of Insecure Libraries"



Let’s find out using FOSS(S) services!



Screenshot from BLACKDUCK

- Bill of Material (BoM) with FOSS components and their versions is generated per each project
- Components with known security vulnerabilities are signaled, with their corresponding level of risk
- The resulting risk is assessed against the acceptable risk threshold (the policy)
- There is a formal process of application security risk management, protecting the developers

Vulnerabilities are detailed for each component

OpenSSL ▸ 0.9.5a

Versions: 275 | Phase: Released | Distribution: External

Displaying 88 Vulnerabilities for OpenSSL 0.9.5a

Filt

	Identifier	Published	Base Score	Exploitability	Impact	Status
›	NVD CVE-2009-3245	Mar 8, 2010	10	10	10	New
›	NVD CVE-2016-2108	Dec 28, 2016	10	10	10	New
›	NVD CVE-2016-2109	Dec 28, 2016	7.8	10	6.9	New
›	NVD CVE-2006-2940	Oct 2, 2006	7.8	10	6.9	New
›	NVD CVE-2002-0656	Jan 1, 2004	7.5	10	6.4	New
›	NVD CVE-2002-0655	Jan 1, 2004	7.5	10	6.4	New
›	NVD CVE-2010-4252	Dec 7, 2010	7.5	10	6.4	New
›	NVD CVE-2016-6303	Feb 23, 2017	7.5	10	6.4	New
›	NVD CVE-2016-2182	Feb 23, 2017	7.5	10	6.4	New
›	NVD CVE-2016-2177	Feb 23, 2017	7.5	10	6.4	New
›	NVD CVE-2012-2110	Oct 7, 2013	7.5	10	6.4	New
›	NVD CVE-2010-0742	Jun 4, 2010	7.5	10	6.4	New
›	NVD CVE-2014-3567	Jul 8, 2016	7.1	8.6	6.9	New
›	NVD CVE-2015-0209	Mar 20, 2015	6.8	8.6	6.4	New

88
Vulnerabilities

Description

The OpenSSL Project is a collaborative effort to develop a robust, commercial-grade, full-featured, and Open Source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols as well as a full-strength general purpose cryptography library. The project is managed by a worldwide community of volunteers that use the Internet to communicate, plan, and dev

Show Less

Released on Apr 3, 2000

Licenses

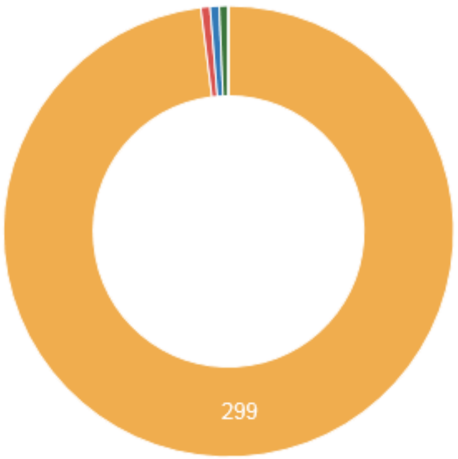
OpenSSL Combined License — Permissive

SSLeay License — Weak Reciprocal

Screenshot from BLACKDUCK

Vulnerability Management is a… managed process

Remediation Status



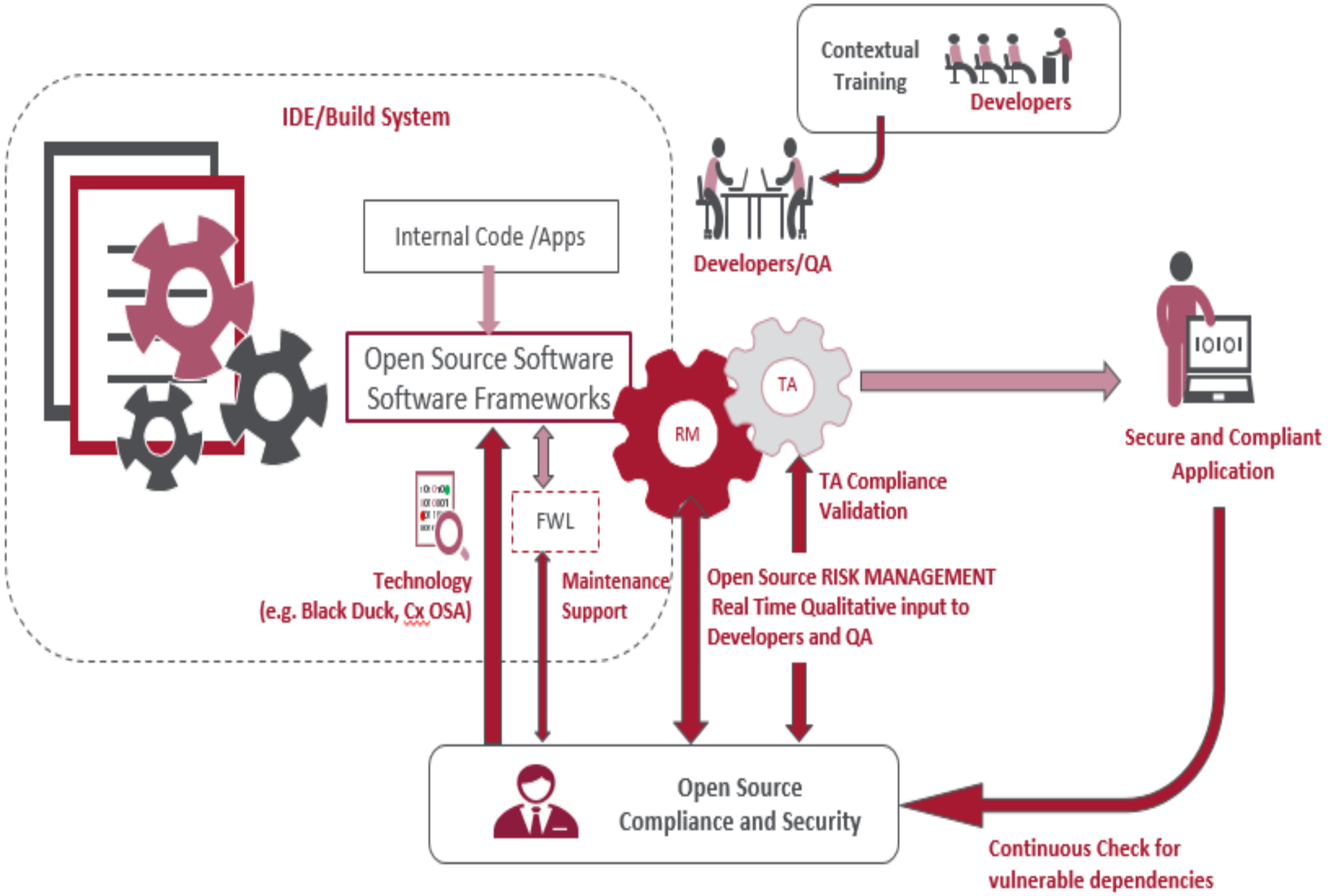
- ☑ New — 299
- ☑ Remediation Required — 2
- ☑ Mitigated — 2
- ☑ Remediation Complete — 2
- ☑ Patched — 0
- ☑ Ignored — 0

Filter vulnerabilities...

Identifier	Published	Aff. Versions	Base Score	Exploitability	Impact
NVD CVE-2016-3082	May 17, 2016	3	10	10	10
NVD CVE-2013-4316	Dec 7, 2016	2	10	10	10
NVD CVE-2017-5638	Mar 27, 2017	2	10	10	10
NVD CVE-2012-0838	Mar 5, 2012	1	10	10	10
NVD CVE-2016-0799	Dec 28, 2016	1	10	10	10
NVD CVE-2016-0785	Apr 12, 2016	3	10	10	10
NVD CVE-2016-2842	Jan 26, 2017	1	10	10	10
NVD CVE-2009-3245	Mar 8, 2010	1	10	10	10
NVD CVE-2016-2108	Dec 28, 2016	2	10	10	10
NVD CVE-2016-0705	Dec 28, 2016	1	10	10	10
VulnDB 145647	Oct 14, 2016	1	10	10	10
VulnDB 103918	Mar 4, 2014	1	10	10	10
NVD CVE-2016-3081	Nov 1, 2016	3	9.3	8.6	10
NVD CVE-2013-2251	Mar 31, 2016	2	9.3	8.6	10
NVD CVE-2013-2135	Jul 17, 2013	4	9.3	8.6	10
NVD CVE-2013-2134	Jul 17, 2013	4	9.3	8.6	10

Screenshot from BLACKDUCK

The holistic process around FOSS(S)Security



- Technical integration
- Processes integration
- IP rights analysis
- Enhancing SAST

Additional security layers might help to see how FOSS components behave in operation!

Network Concerns   Ted Guild, W3C Automotive Lead

Genivi AMM, Birmingham, UK

May 10, 2017

Attack Surface Size

Vehicle's interet connection is the biggest attack surface

- Reaction from technical peers
- Internet is a hostile environment

Genivi Security Expert Group

Sound Practices

- Education
- Coding Guidelines
- Code Analysis
- Threat Modeling
- Architecture
- Layering
- ...

High Level

- Connection Accounting
- External Site Security Evaluation
- Imposing Rigid Network Access
-
- Guidelines

Ecosystem

We are building a framework for 3rd party apps (FB, Waze, Pandora)

- HTML5/QT/Headless
- Signals
- Nav/LBS
- Media (services, library)
- Notifications
- Payments
- Wishlist: traffic, weather, speech…

Other day job - Head of IT

- Domain hijacking
- DDoS
- State actor probes
- Phishing
- Misuse of services we provide
- Code Audits
- Penetration Testing
- Compromise Forensics
- Counter measures

Memories...

Remember when you could account for every network connection?

- IP logger in simpler times
- CDN, trackers, adverters, "like us"
- Blockers: Flash, Ads, Trackers, JS

Development and Testing

- Know every connection you require
- Run traffic monitors during testing phases
- W3C's DTD traffic problem

Lock it down

Possible package requirements for 3rd party apps. Suggestions partially address OWASP top ten

- DNS zone files
- Accompanying Firewall rules
- Apparmor/SELinux/Smack rules
- Static SSL Certificates in /etc/ssl/certs
- All Javascript permitted to run should be packaged not fetched
- Might as well package all needed images, css, html etc

SSL hardening

merely using SSL alone is not enough

- HTTP Public Key Pinning (HPKP)
- Cert strength requirement - beyond merely no SHA1
- Site SSL evaluation tools
- HTTP Strict Transport Security (HSTS) & Upgrade Insecure Requests (UIR)
- Content Security Protection (CSP)
- Follow W3C's WebAppSec WG

Web Application Firewall (WAF)

Another idea

- Web Application Firewall - car does its own MiTM to outside world
- Can run same or another WAF as security layer to Web Sockets and HTTP REST services on vehicle
- Apache mod_security example
- Limit methods (GET POST), inspect permitted parameters
- restrict which apps (token or other id)
- Control what data is allowed to leave the vehicle

Web Application Firewall (WAF)

Continued

- These rules for a given 3rd party app could again be part of package
- Limit content types - no Javascript from outside world
- Verify content types, ensure no injection of malicious (eg tainted media files)
- Sensitive needed content can be signed with W3C WebCrypto API
- It can cache content too, useful for intermitten connectivity and performance

Open Browsing

Hearing some are considering allowing full open browsing from vehicles

- WTF, seriously?
- Yes the guy from W3C is saying he doesn't want your car on the web - personal opinion
- Sales decision
- Fine, put it in its own vm
- Zero connection capabilities to APIs being exposed
- Immutable/Read-only FS or clean image on each reboot?

Feasability?

- Fragmented industry / multiple platforms
- Marketing & Business forces
- W3C Guidelines
- Genivi Platform implementation
- Get Involved!

Thank You

- Questions
- Follow up: ted+auto@w3.org
- https://www.w3.org/auto

Software Security

May 11, 2016 | Overview

Stacy Janes Chief Security Architect - Irdeto

Assaf Harel CTO & Co-Founder – Karamba Security

Security Team, GENIVI Alliance

Software Security 101

Integrity and Confidentiality

Integrity → Proving the validity of data. → Digital Signature

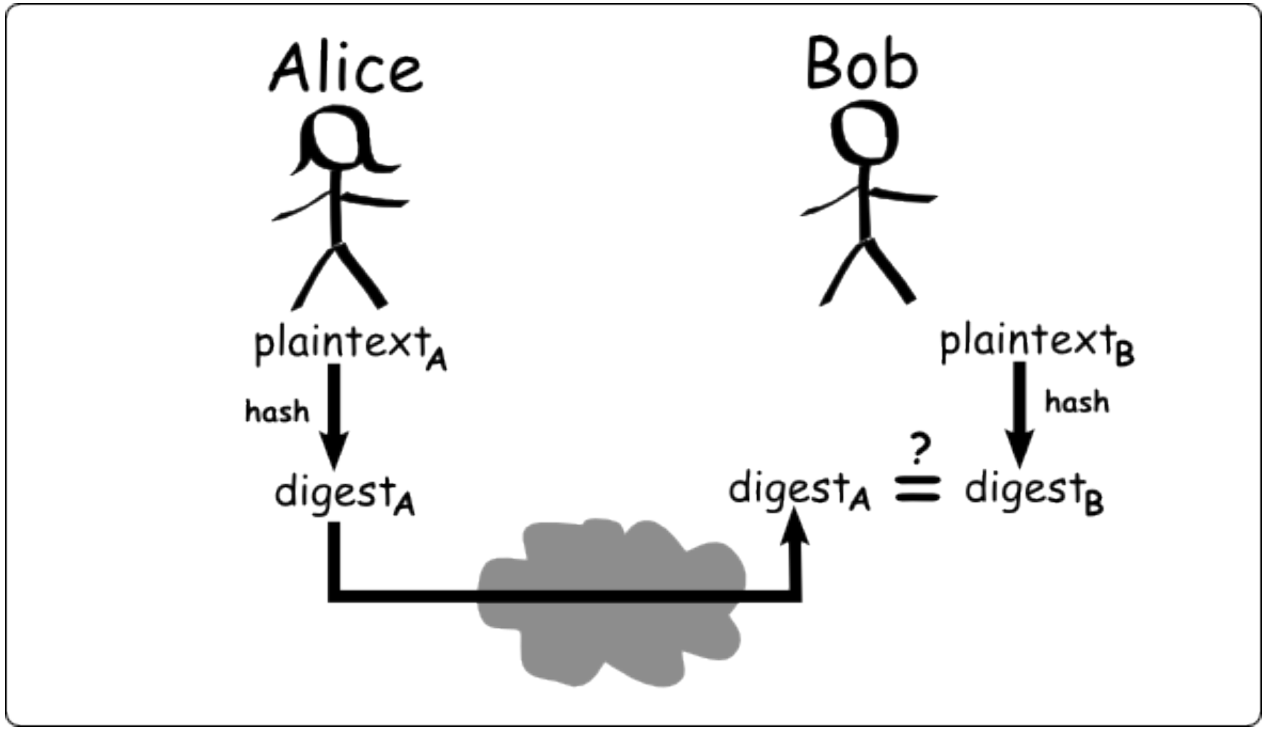
Confidentiality → Protecting the contents of data. → Encryption

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 and not encrypted when stored.

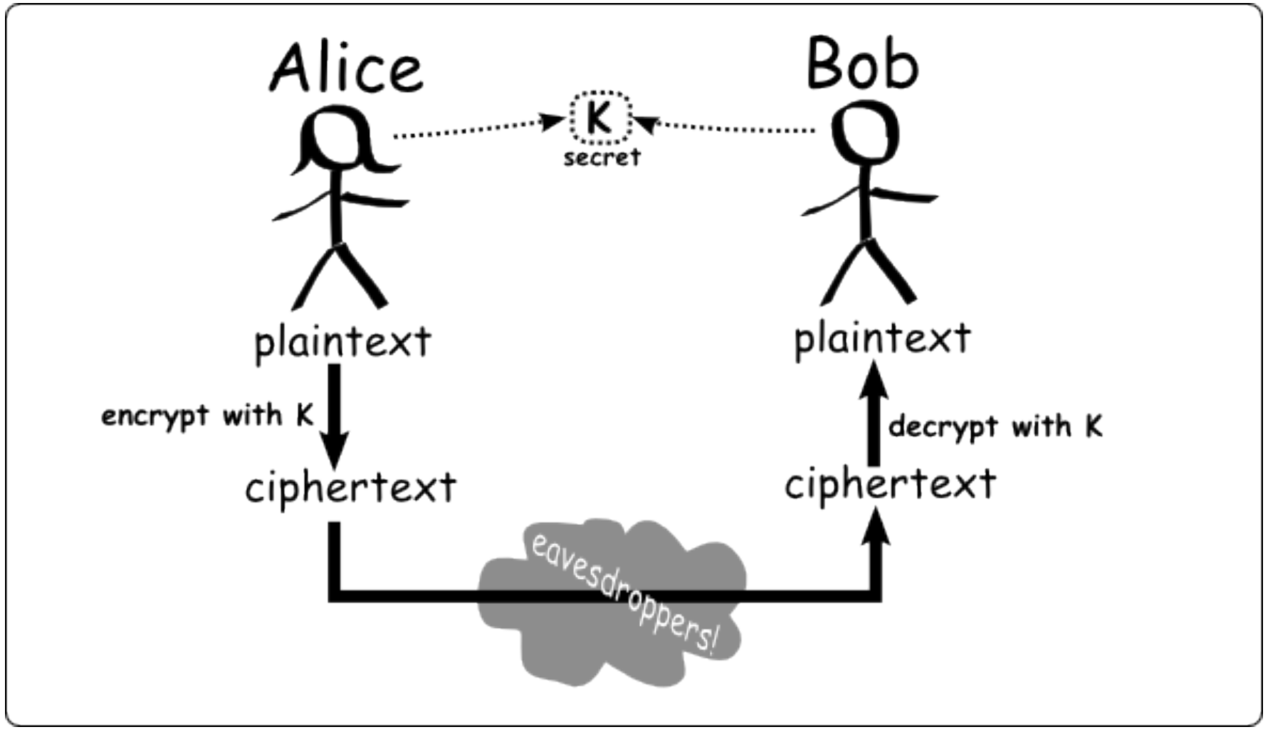


Encryption – Symmetric Key

Encryption and decryption done with the same key

Symmetric cryptography is fast (relative to Asymmetric)

Key management becomes cumbersome beyond a few actors.



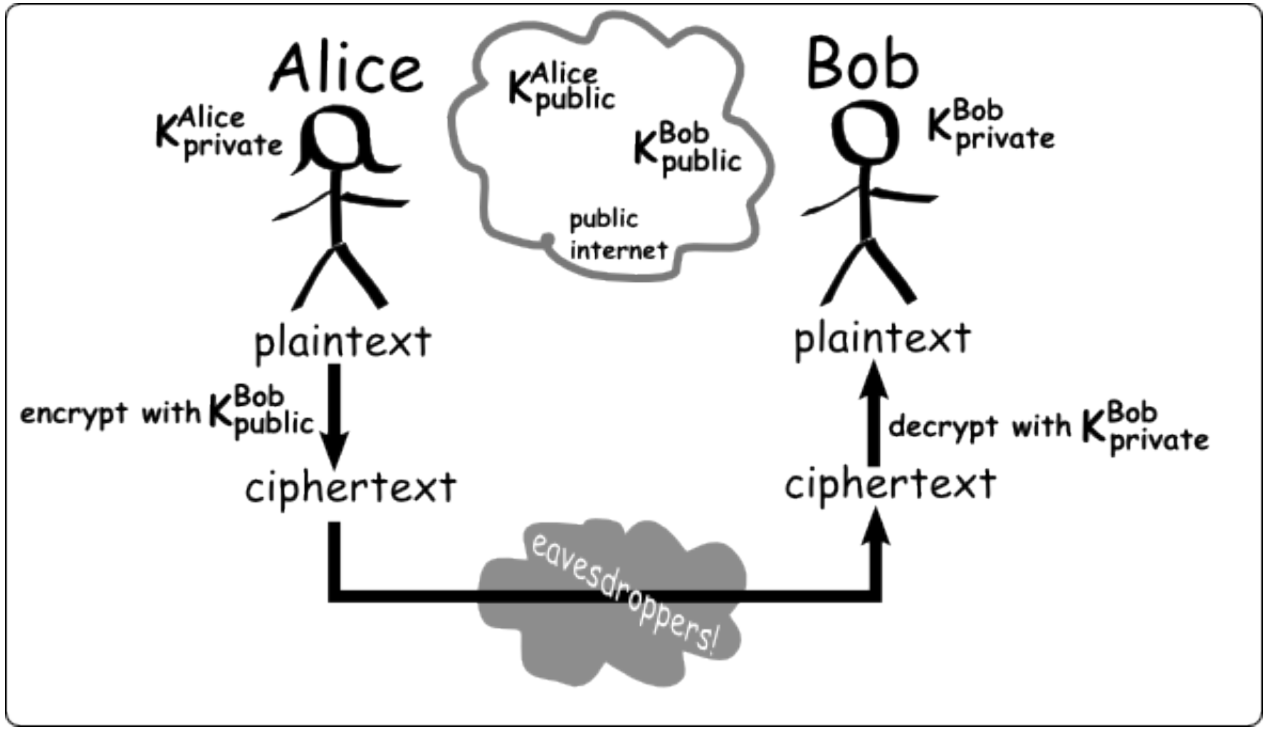
Encryption – Asymmetric Key

- ➢ 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 enable this.



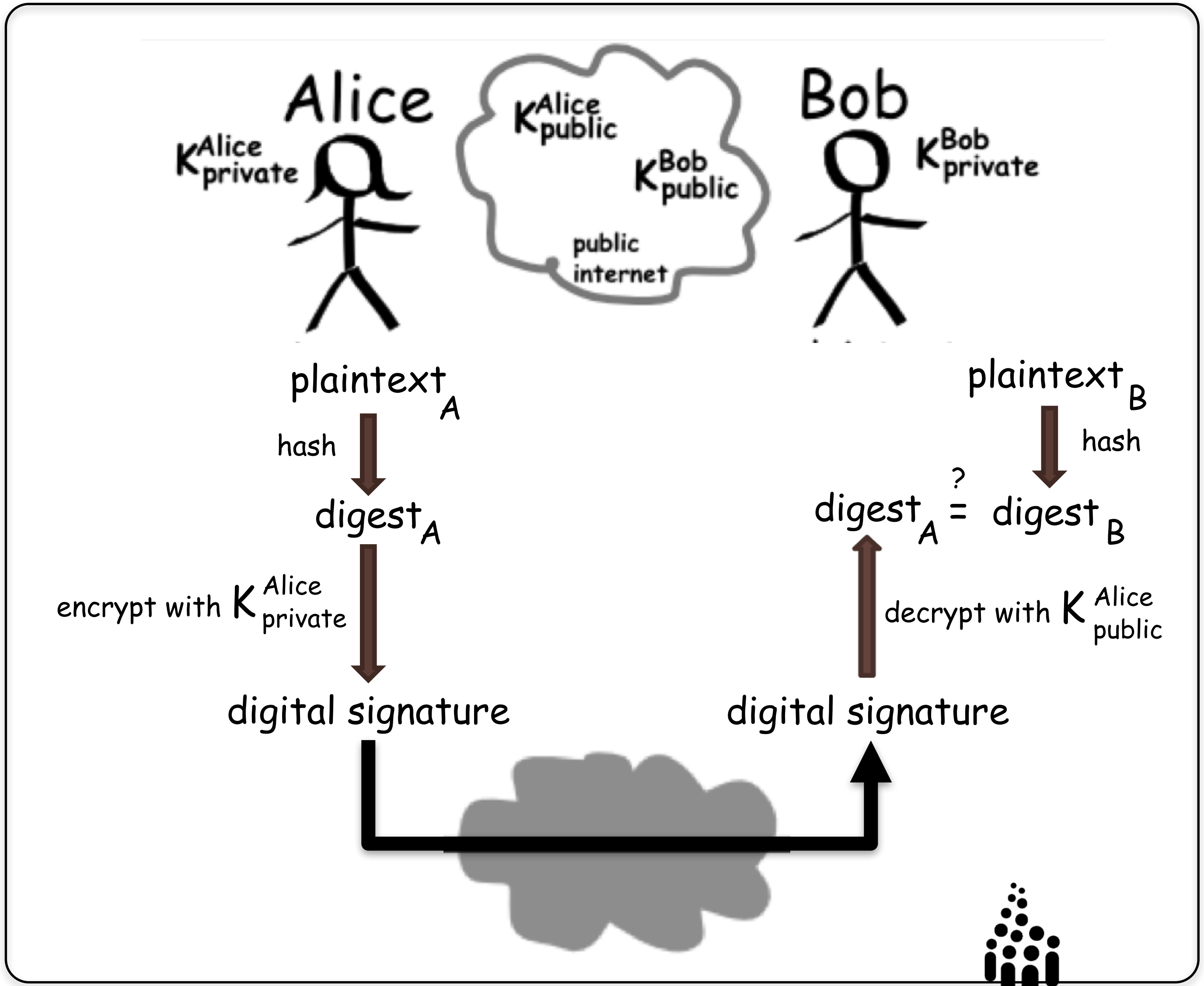
Digital Signature

Encrypted Hash

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

X.509 Certificate around Public Key for identity verification

Does not hide data



“Defeating” Crypto – Easier to Bypass

Brute force is typically not a realistic attack

End point access opens up attack vectors

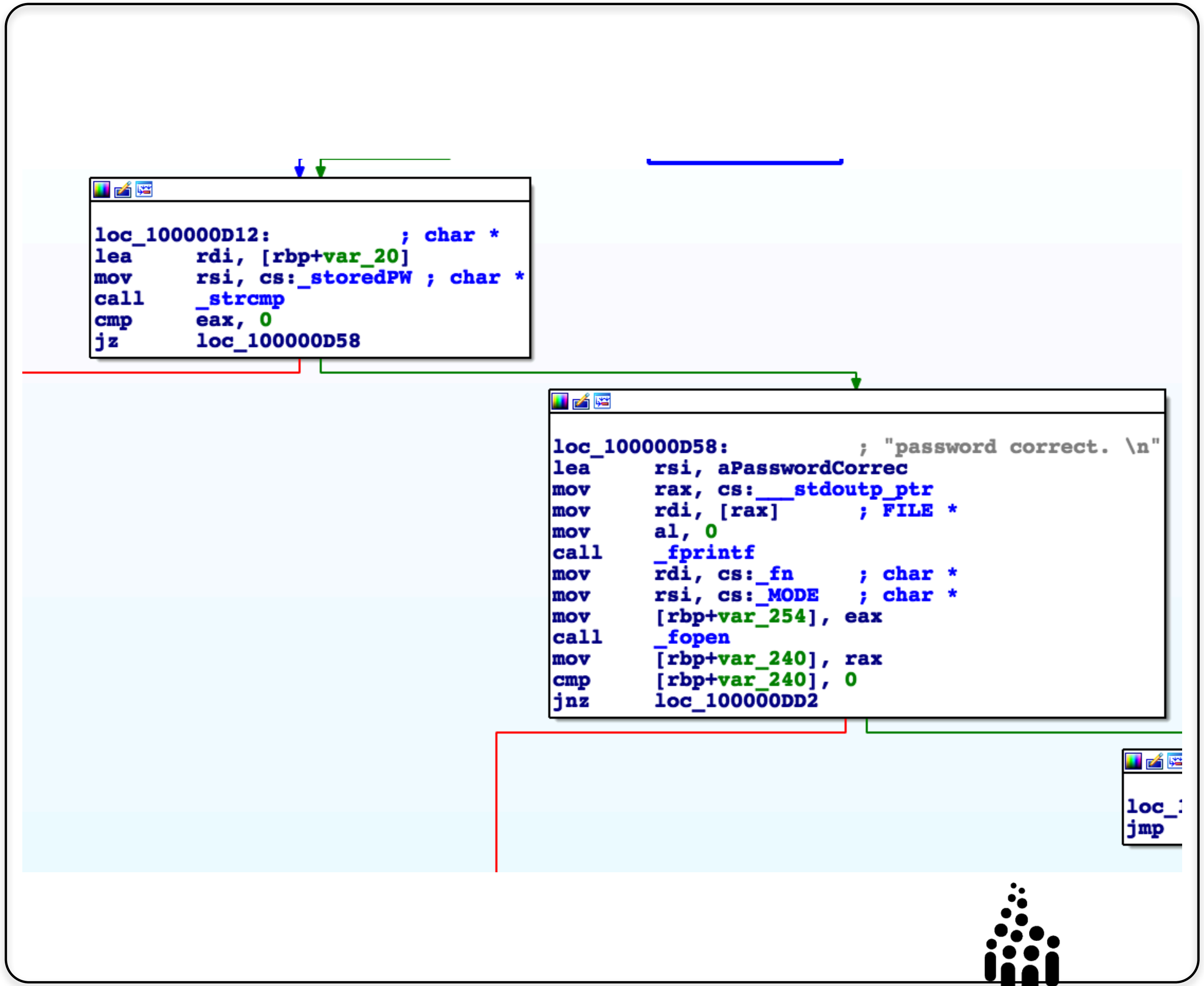
- Key lifting. Easy for software key if not properly protected
- Binary modification to “jam” logic branch for signature check
- Lifting clear data from memory after decryption
- Inserting malicious data to be signed/encrypted
- Shimming interfaces

Branch "Jamming"

Let software verify signature

Find branch that checks return code

Reverse comparison opcode to allow invalid signature to pass



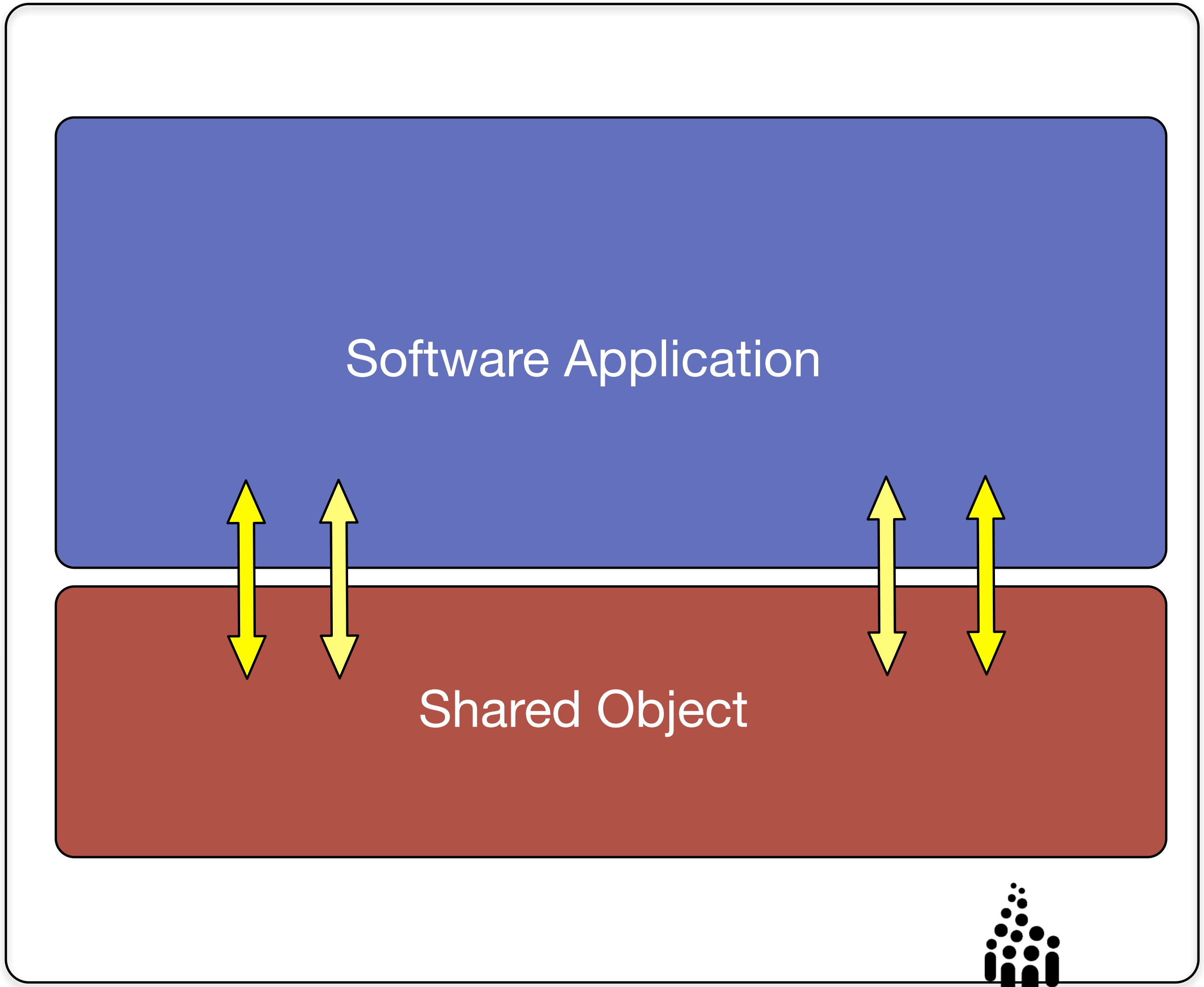
GENIVI®

"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.

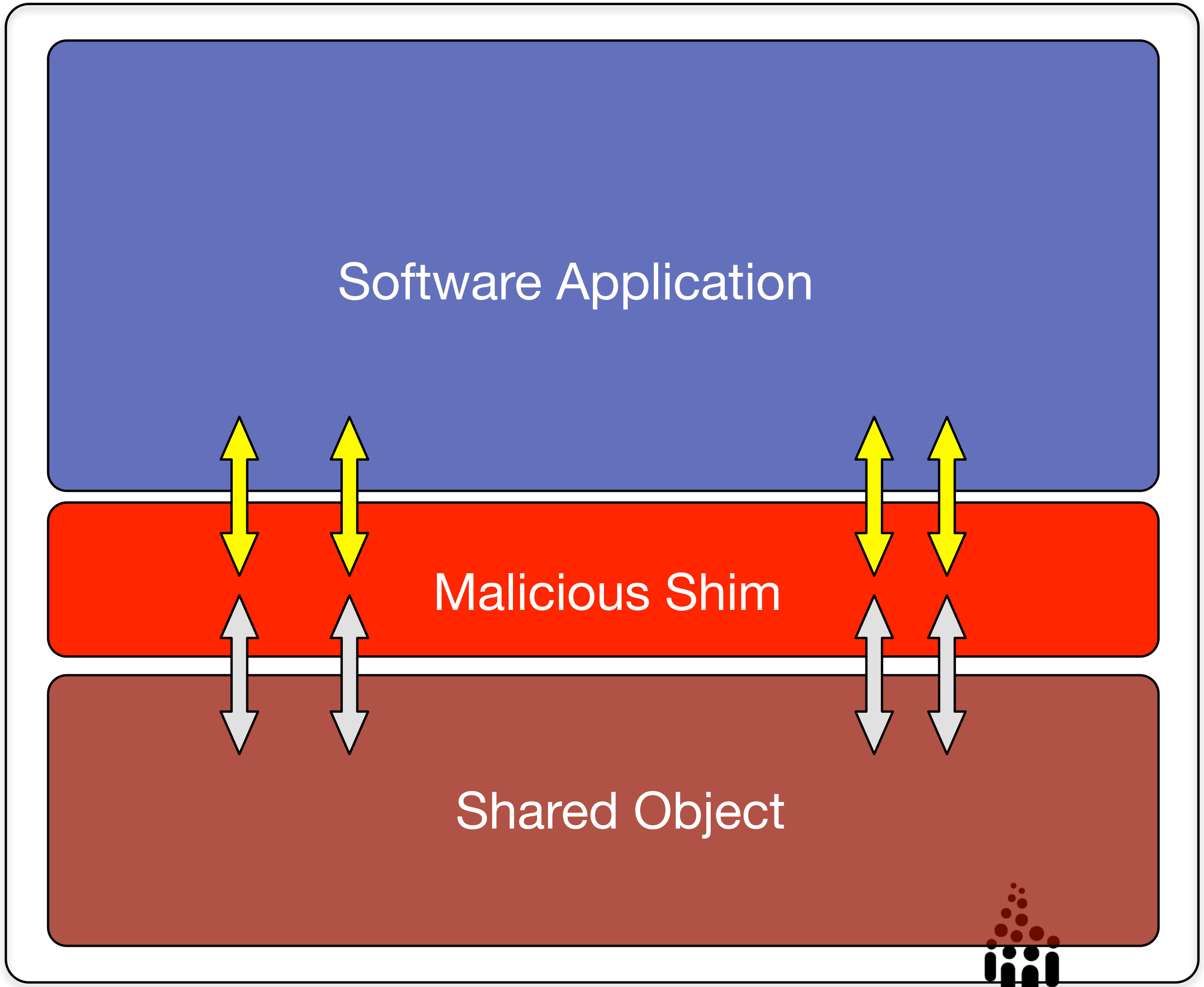


"Shimming"

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.



Software Protections – Integrity Verification

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 - Obfuscation

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

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

Obfuscation of open source can be tricky. License issues. Leakage of information through system calls.

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.

Assertion Check

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

Entangled

```
pwHash = getPasswordHash();
decryptFile(fn, pwHash);
```

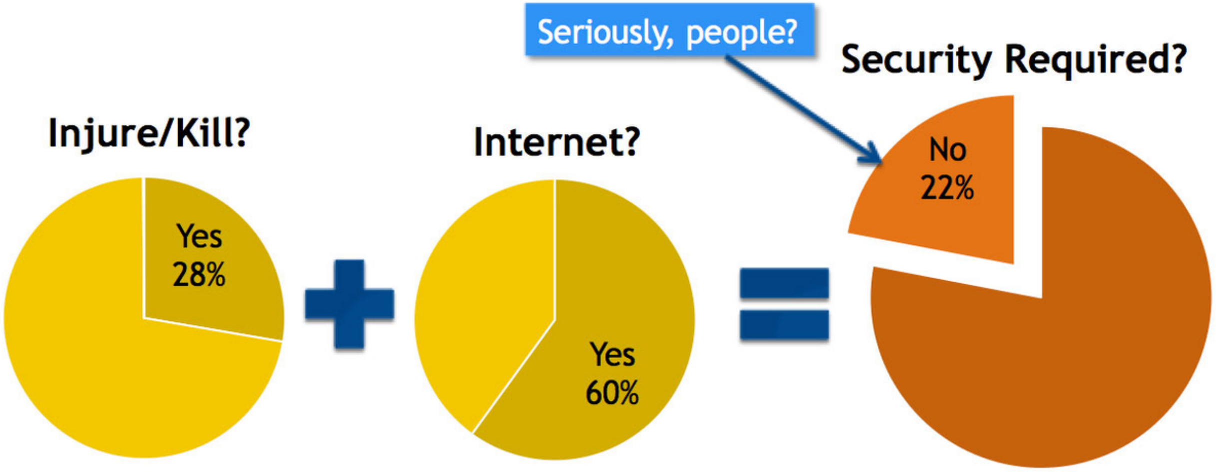
Platform Security Architecture

May 11, 2017| Hardware Enforced Security for Automotive

Erik Jacobson

Marketing Director, ARM Architecture Technology Group

Security: the climate change of engineering



Source: Barr Group
2017 Safety and Security Survey

Security is critical for connected vehicles

- Costs of non-compliance

  Violations/fines, higher insurance premiums, litigation…

- Regulatory

  Critical Infrastructure Policy (ex. U.S. Federal EO 13636)

  NIST Cybersecurity Framework

- Industry compliance (ex. ISA/IEC 62443:EDSA)

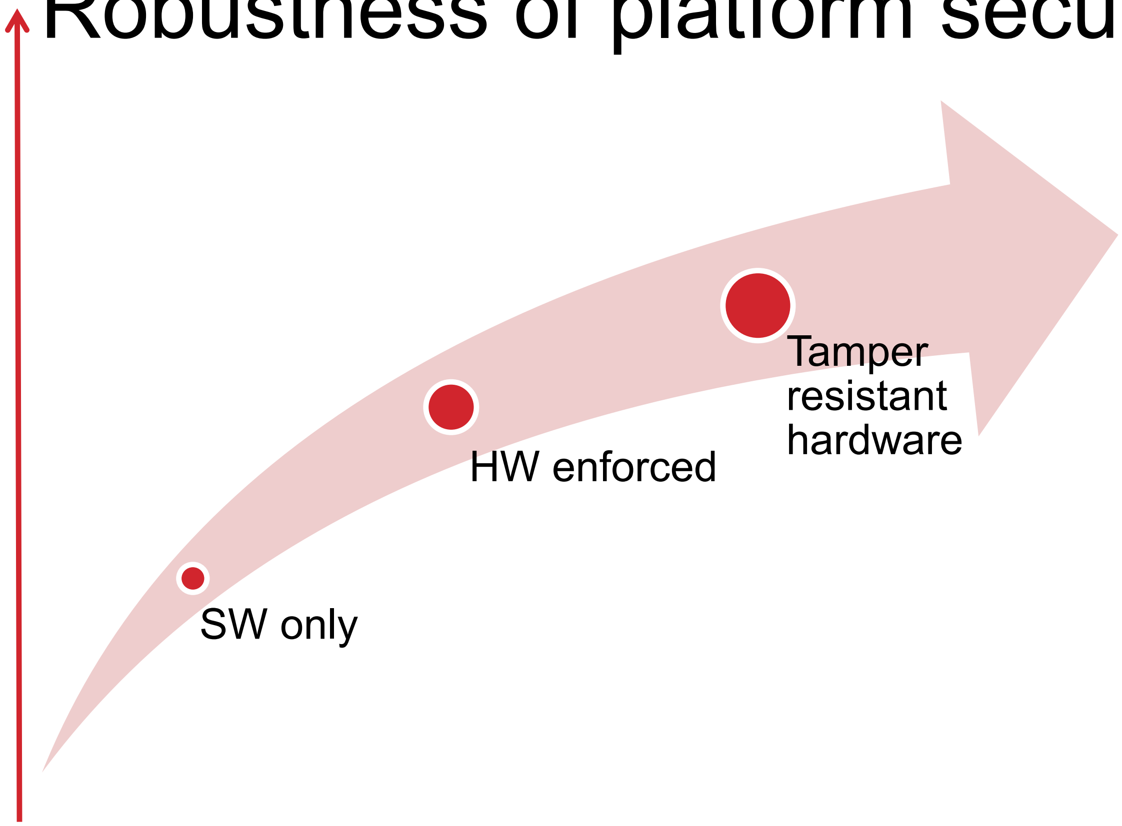
- Market pressures

  Risk management from loss of credibility

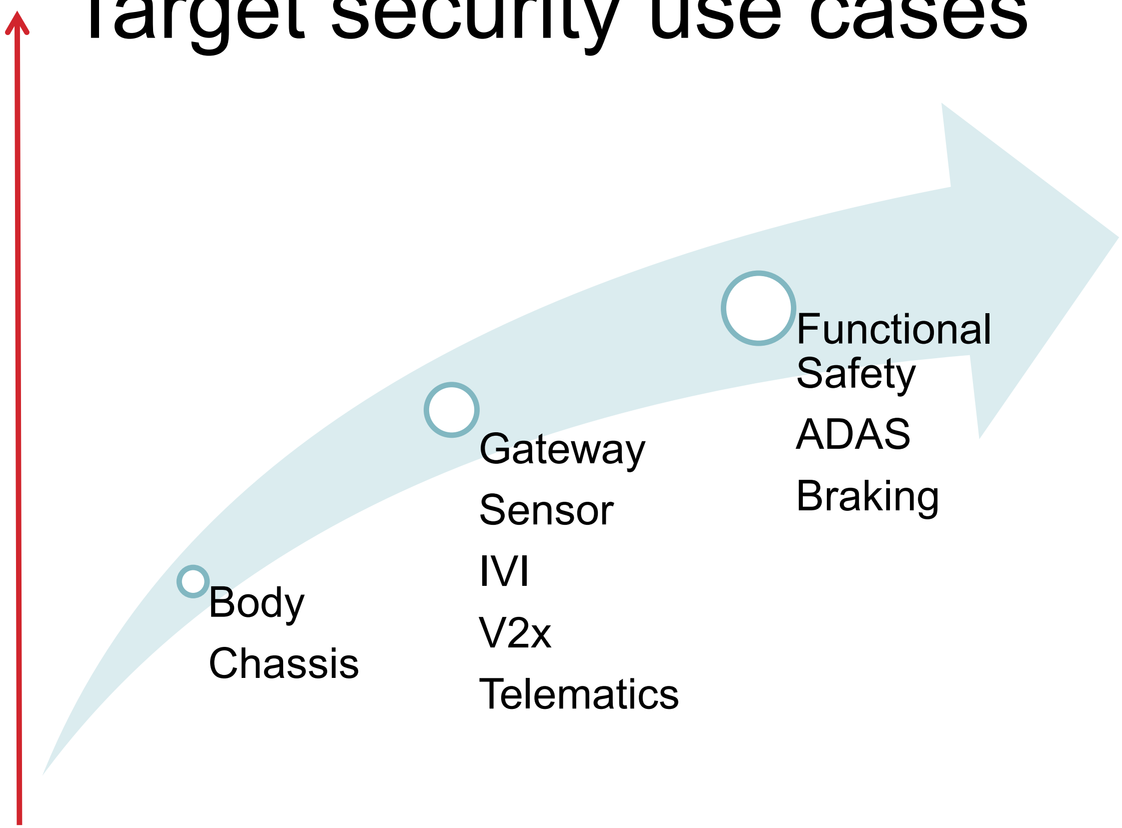
  Loss of proprietary or confidential information/assets

Choose the right-sized security solution

Robustness of platform security

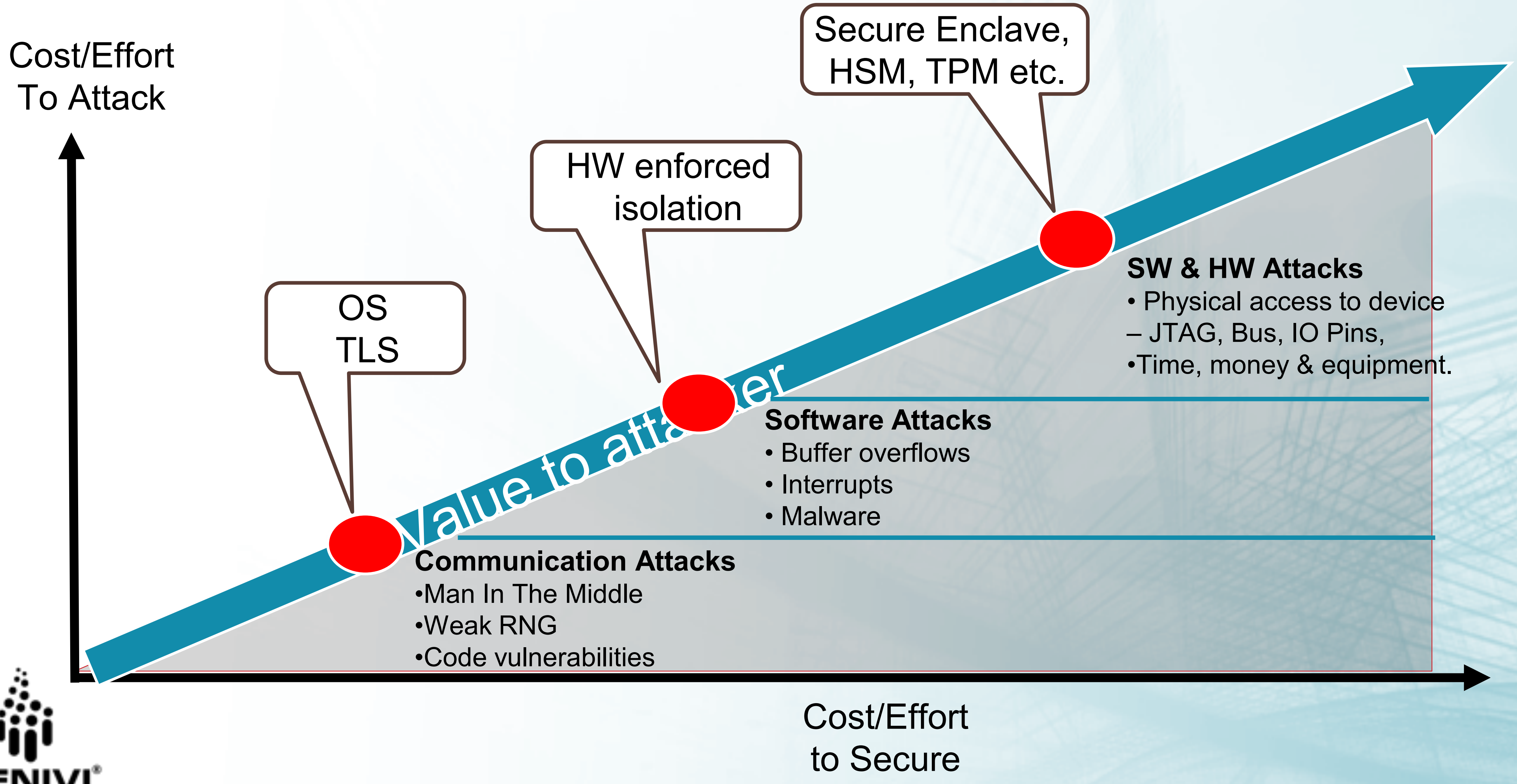


Target security use cases



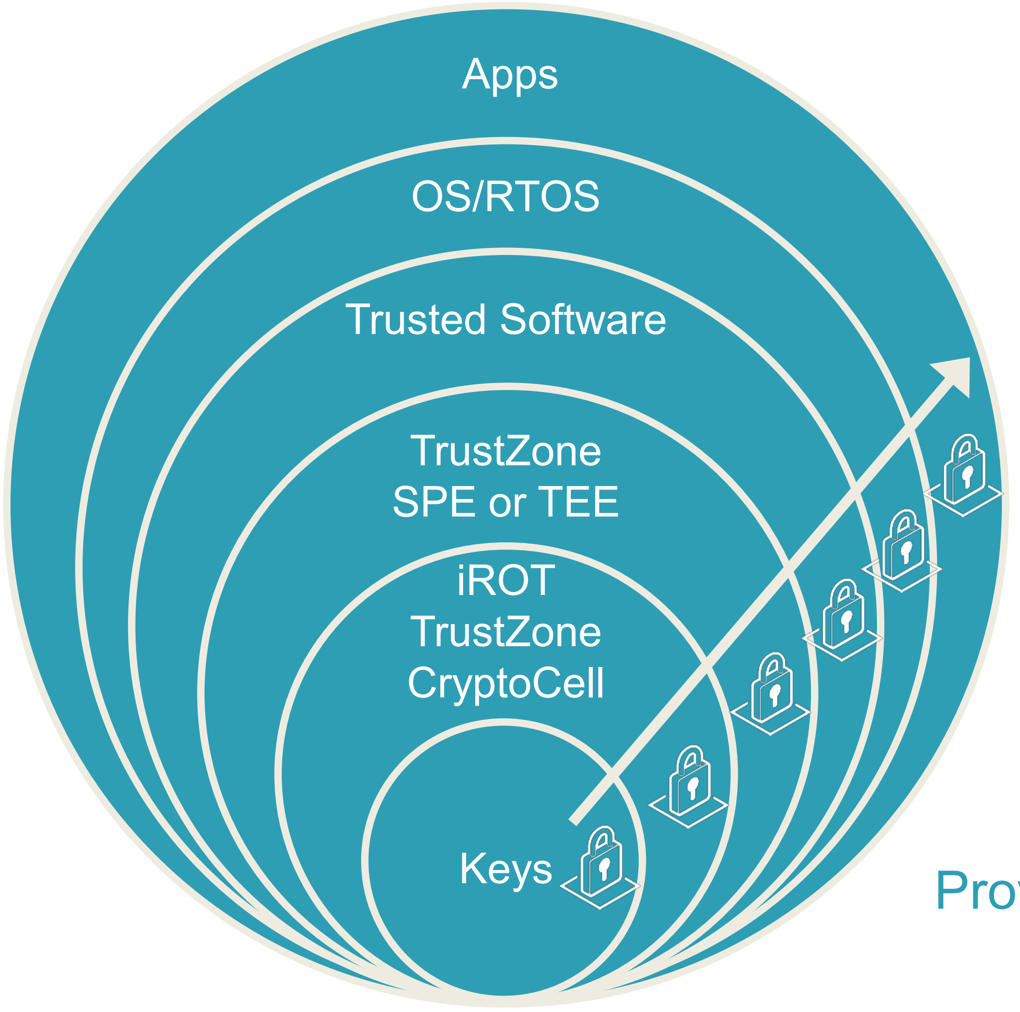
HW enforced security expands implementation options and flexibility while still offering robust architecture options

Security targets for different threats



GENIVI®

Establishing trust and integrity based on hardware



OS/RTOS

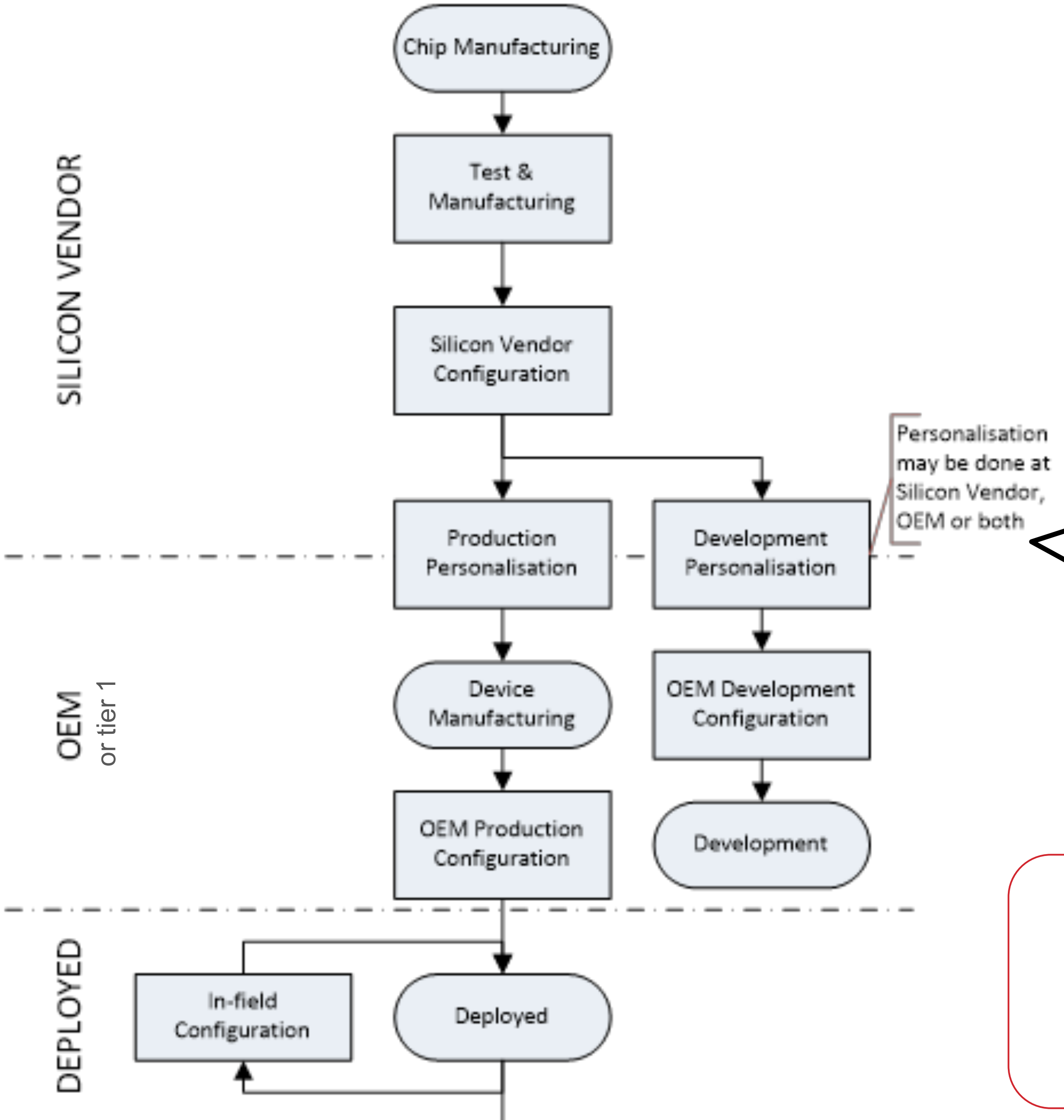
Trusted Apps/Libs

Extended Root of Trust e.g. TrustZone based
TEE or Secure Processing Environment (SPE)

Initial Root of Trust: Dependable Security functions

Provisioned keys/certs

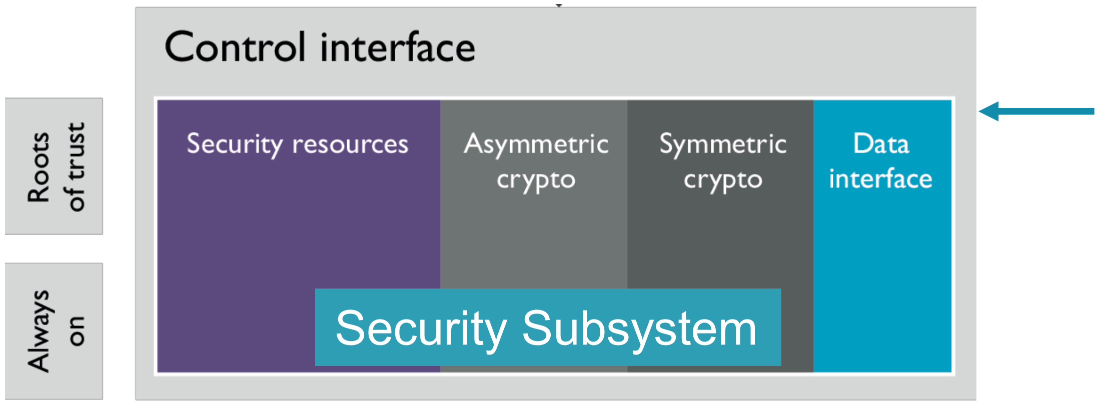
A Root of Trust starts at manufacturing time



Injection of cryptographic assets such as unique keys

Know your supply chain!

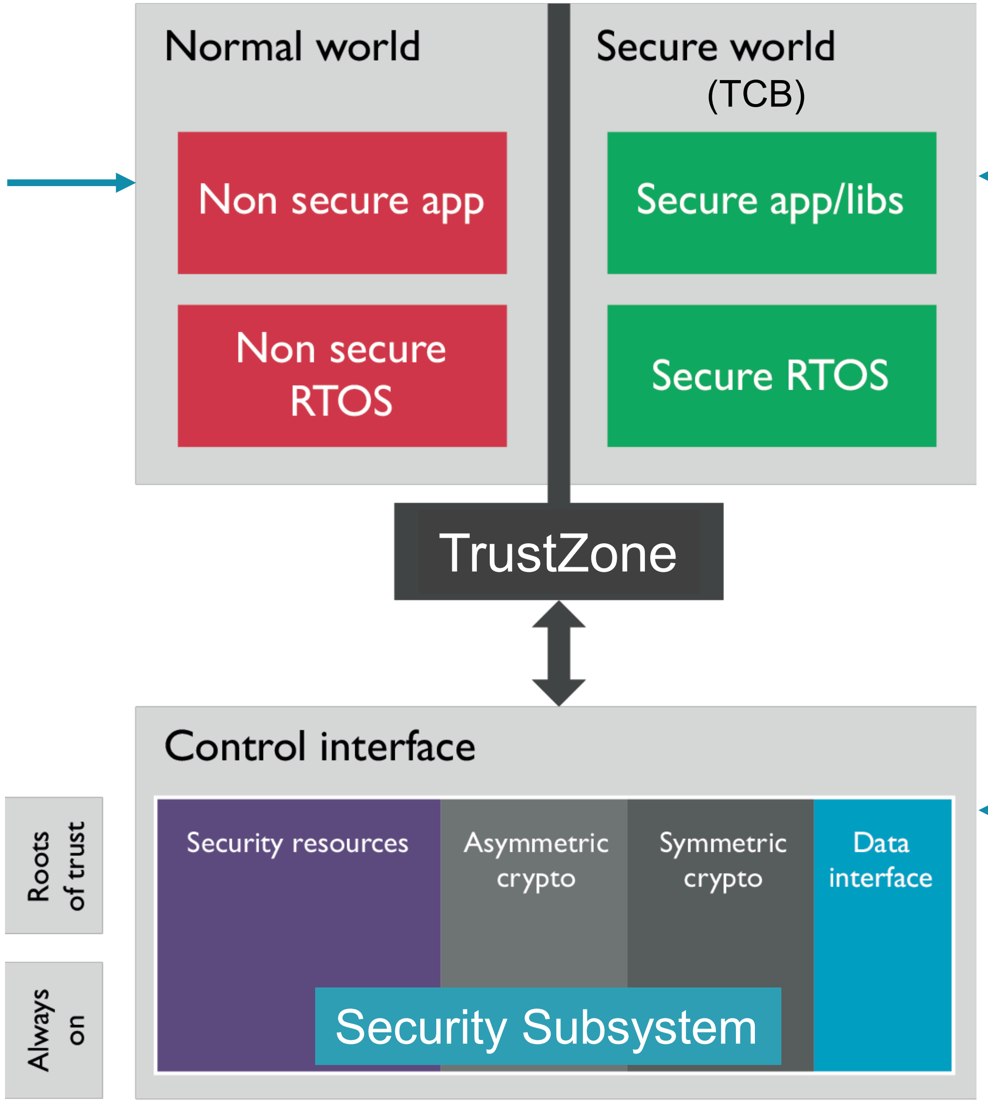
Ideally a RoT lives in a security module…



Security subsystem
Highly evaluated code developed by security specialists & built in by silicon vendor

…and is exposed via hardware-enabled isolation layers

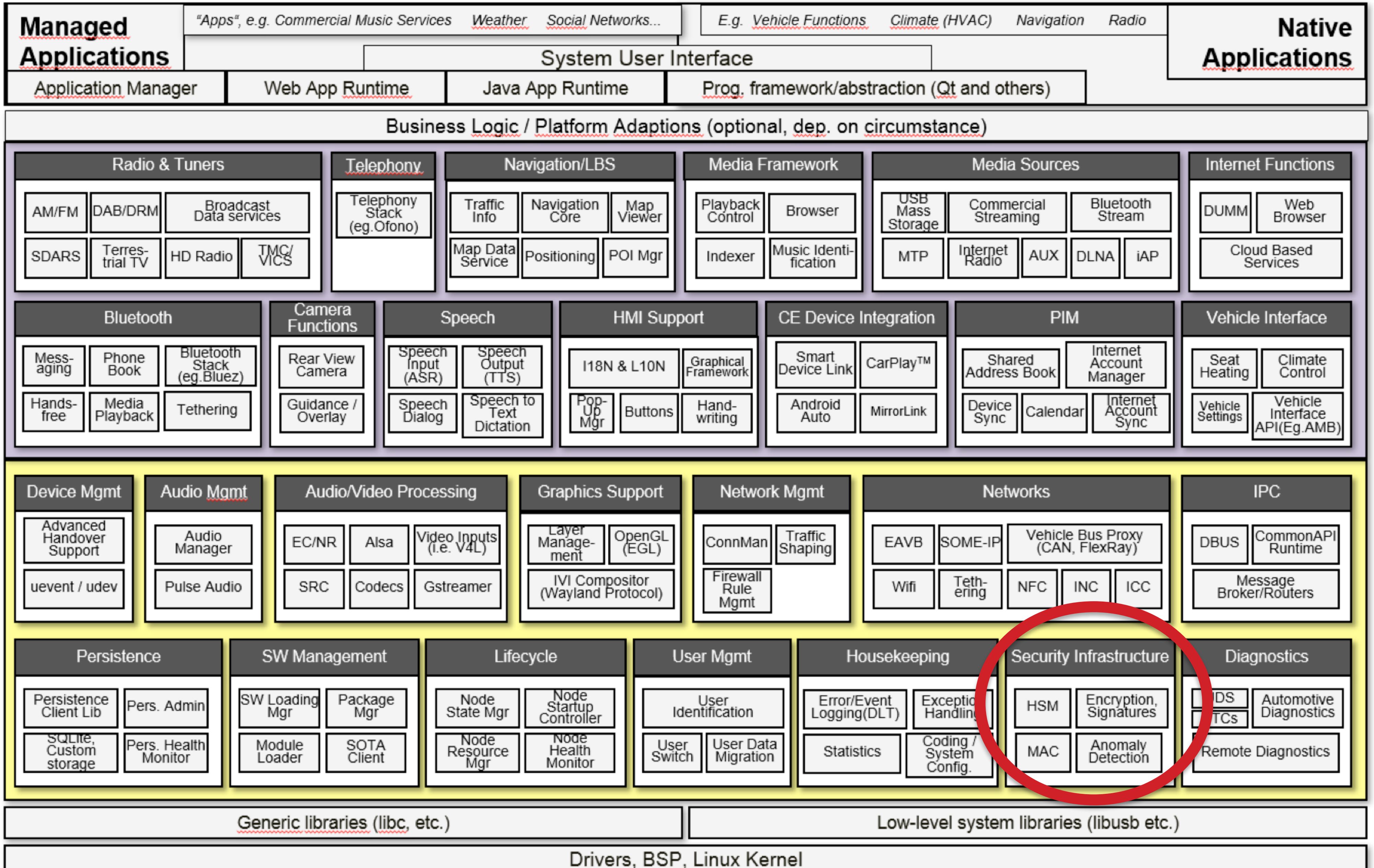
Normal World
IoT developer writes Apps
On top of his/her chosen OS/RTOS



Secure World
= Trusted code (mostly libs)
Often on top of an audited/reviewed RTOS or similar – the Trusted Execution Environment (TEE)

Security subsystem
Highly evaluated code developed by security specialists & built in by silicon vendor

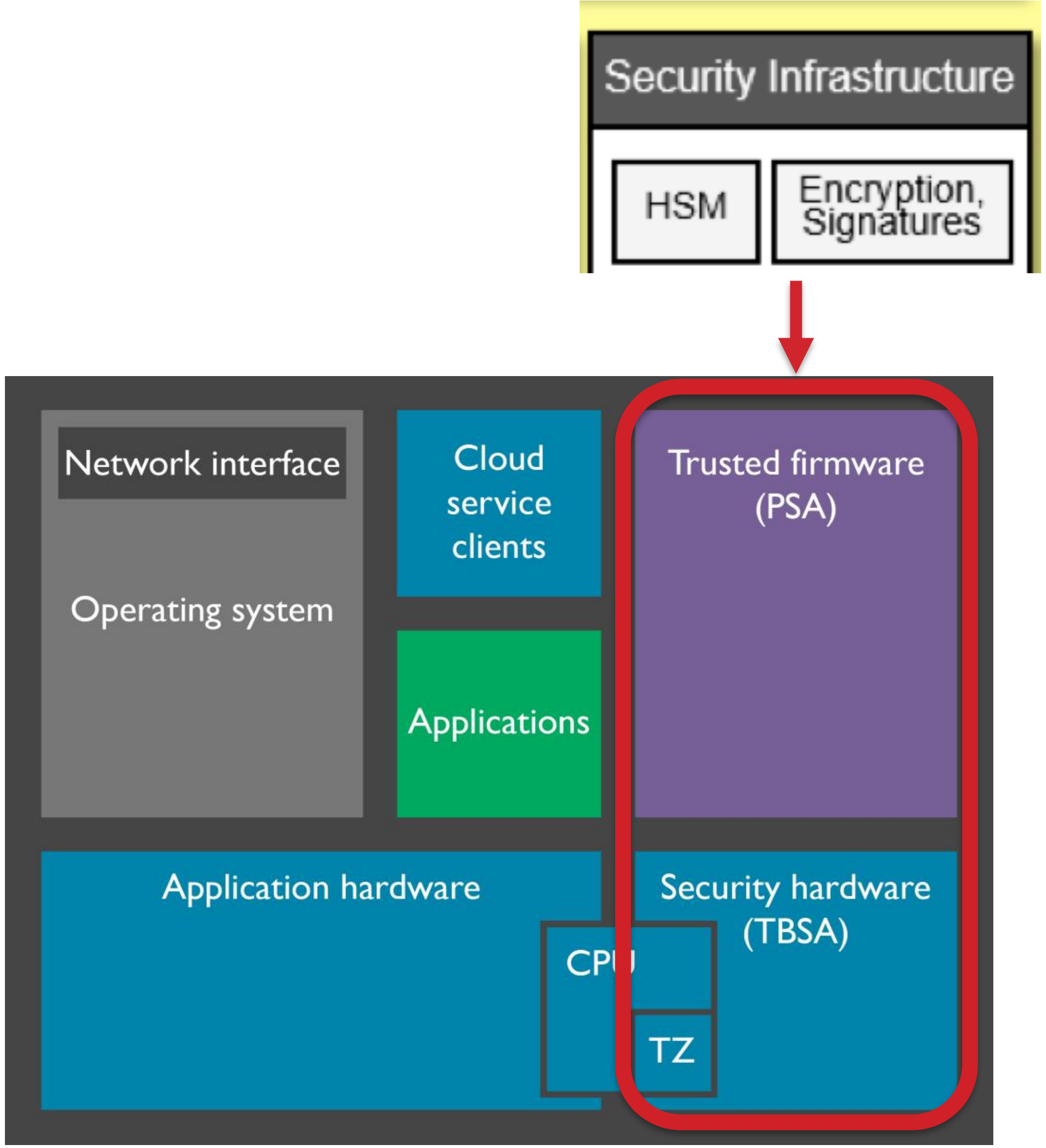
SW needs a std way to access security functions…



… but we need to consider HW-enforced isolation

- To enable HW-enforced isolation, the most sensitive SW modules need to be re-factored between “normal” and “trusted” (secure world)
- This can be time-consuming and involves effort

Architecture standards link the HW & SW communities



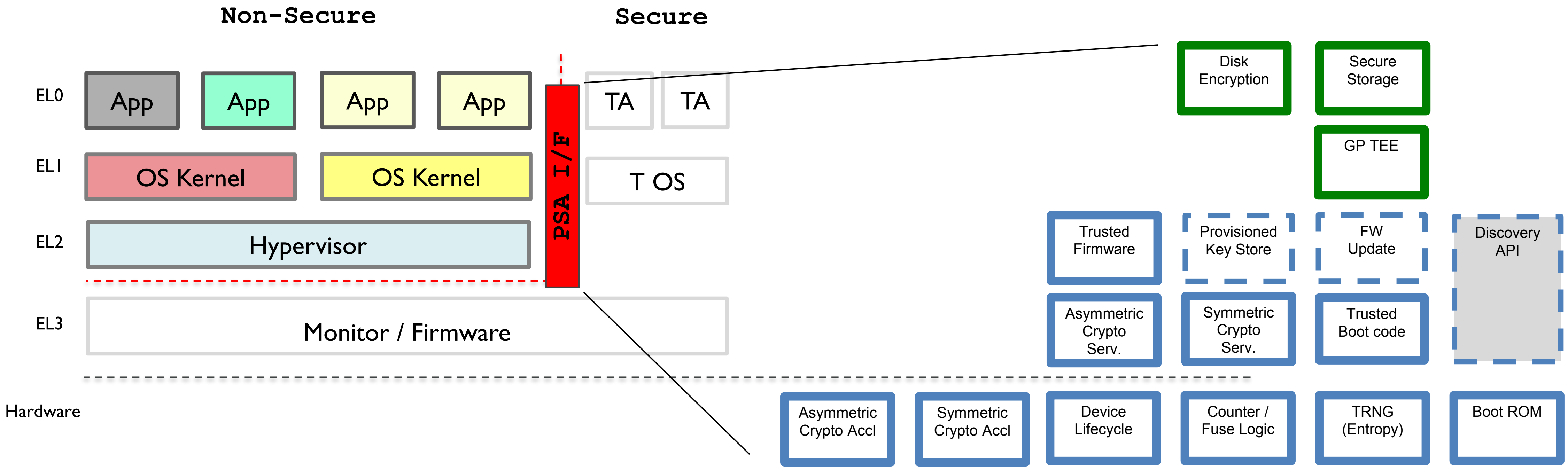
- Track 1: Standardize APIs for the SW community, supporting TEEs and an upcoming Platform Security Architecture (PSA) specification
- Track 2: Guide the SoC design community with Trusted Base Security Architecture (TBSA)
- GENIVI architecture security components can be mapped through TEE/PSA onto TBSA.

Platform Security Architecture (PSA)

GENIVI®

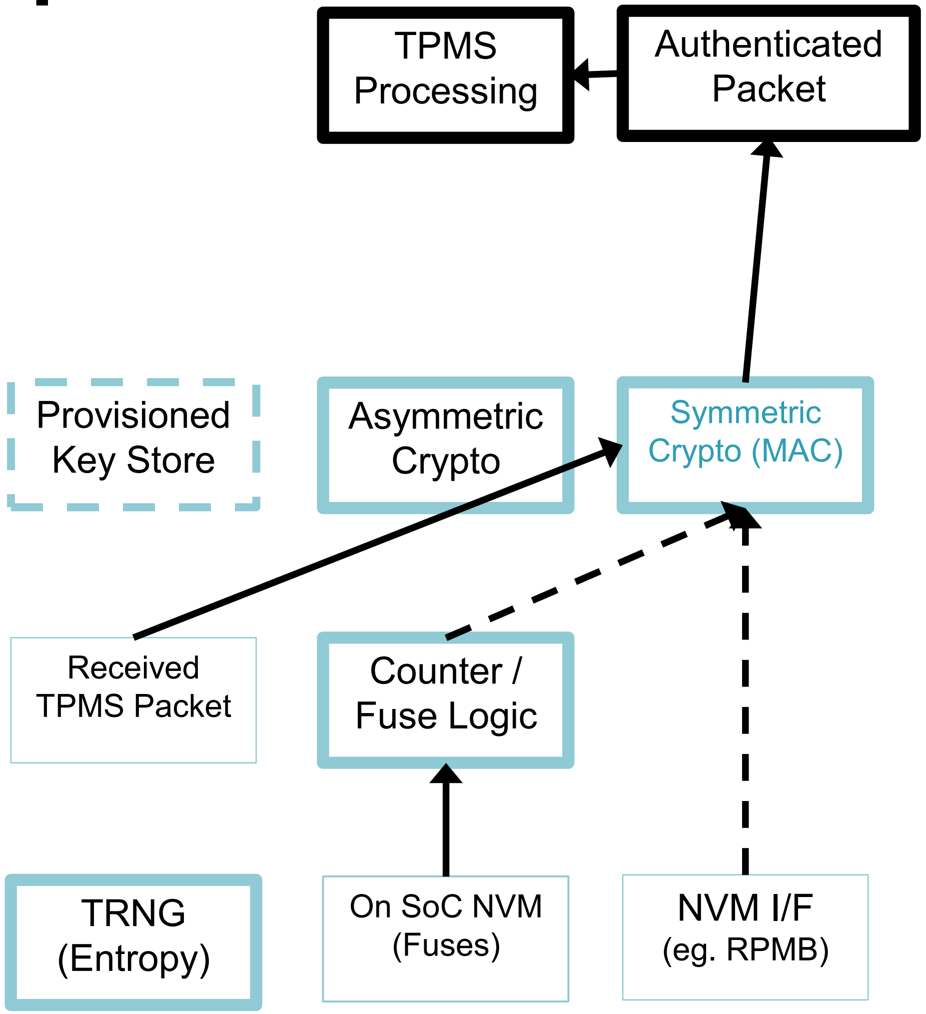
Platform Security Architecture

- ARM hopes to provide an interface to, and generic framework for, the essential secure functional building blocks
- Reference implementations will provide models of how to construct the security system, including integration of ARM security IP

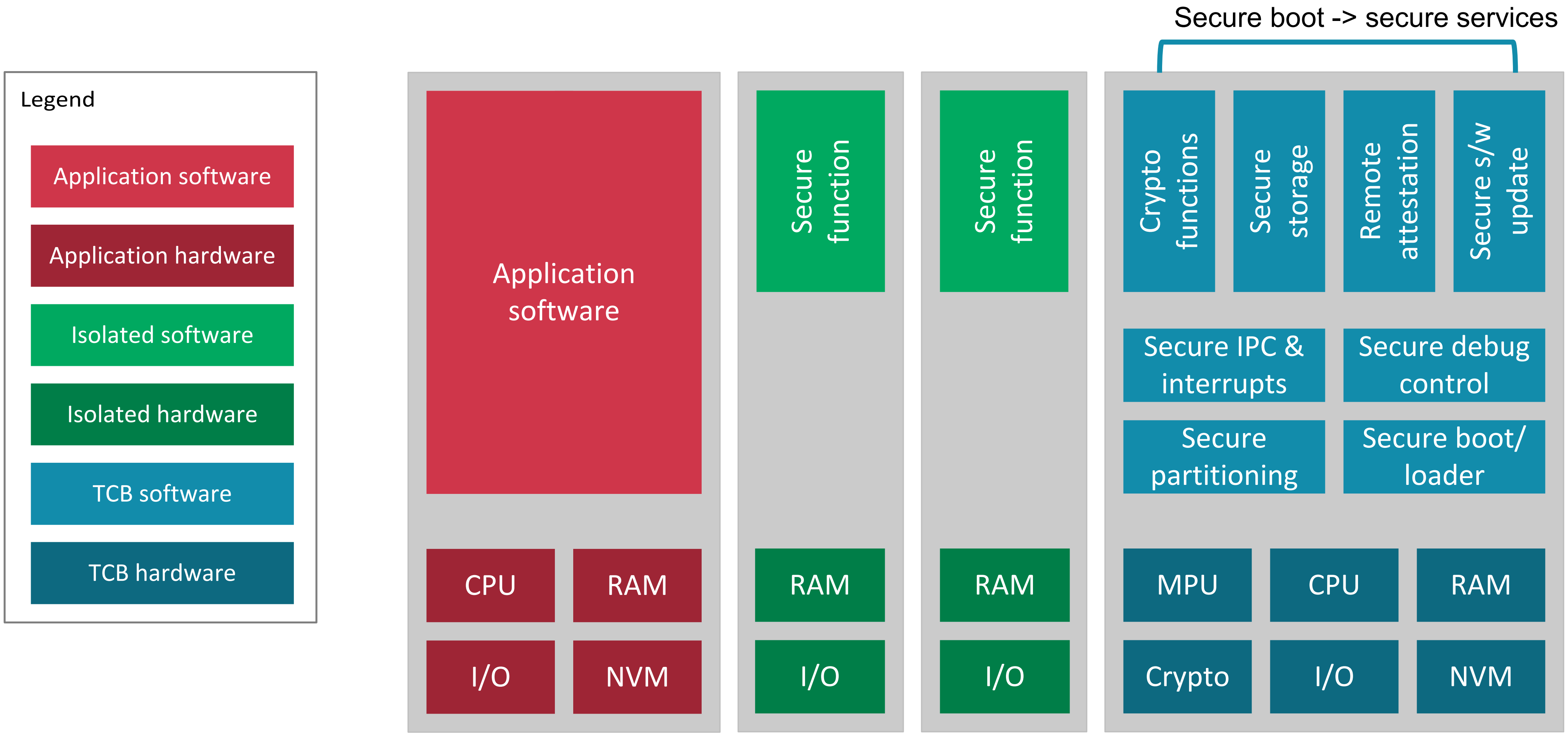


Security use-cases may be systematically decomposed and repeatedly implemented

- TPMS messages can be protected by message authentication code (MAC)
- Normally produced by a keyed cryptographic hash function.
- This protects both message authenticity and integrity.
- TPMS and ECU SoC contain shared secret key
- Key is securely stored and accessed by s/w using PSA
- TPMS packet may containone-time data e.g. sequence # , nonce etc, to protect against replay attacks.



The role of Secure Boot



Summary

- HW enforced isolation is an important next step-up from SW-only security but is below tamper-resistant HW on the attack-value graph
- HW enforced security starts in the factory – know your supply chain!
- API standardisation is important (of course!)
  - But think in 3D not 2D when dealing with secure memory and device HW architectures
  - Take advantage of TrustZone – but takes work to audit & re-factor code
- Platform Security Architecture (PSA) will standardize core secure functions on ARM systems, underneath TEEs where present

GENIVI®

Threat Assessments and Attack Trees

May 2017 | You Can Do This (!)

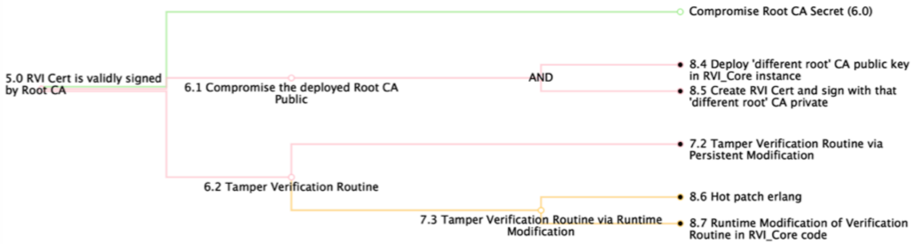
Ben Gardiner

Principal Security Engineer, Irdeto

Agenda, etc.

- 20 minutes
- What are Attack Trees? What are Threat Assessments?
- How can I?
- Should I?

Subtree 5.0 RVI Cert is validly signed by Root CA



5.0 RVI Cert is validly signed by Root CA Attack Subtree

Attack Vector Node 8.4 Deploy 'different root' CA public key in RVI_Core instance

An attacker tampers with a deployed RVI_Core instance's resources where the public key of the root CA is stored. They replace this public key with their own so that they can spoof the authentication server. If they spoof the authentication server then they can generate their own credentials.

Mitigation Required

For Implementors of RVI

Credentials for RVI_Core need to be protected a rest and in memory against tampering (and replacement).

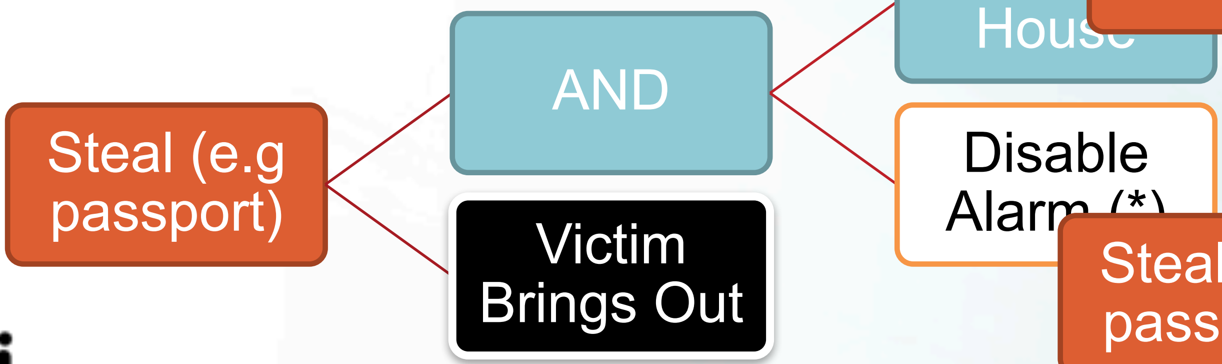
More Effective to Fix Early

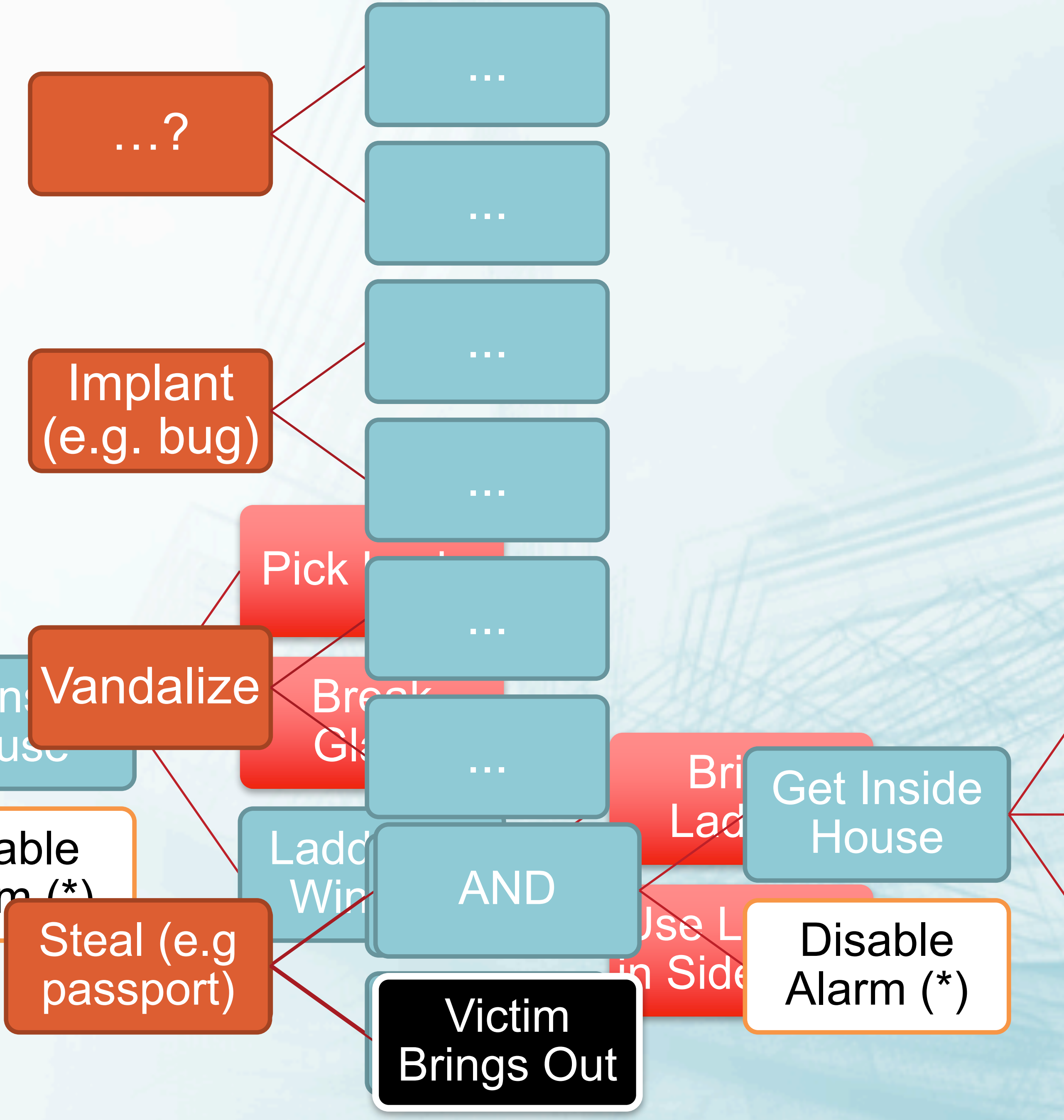
- Fixing Later → Overhauls
- Does take time now, but could save time in the long run
- And focuses efforts to areas that need it

GENIVI®

Attack Trees?

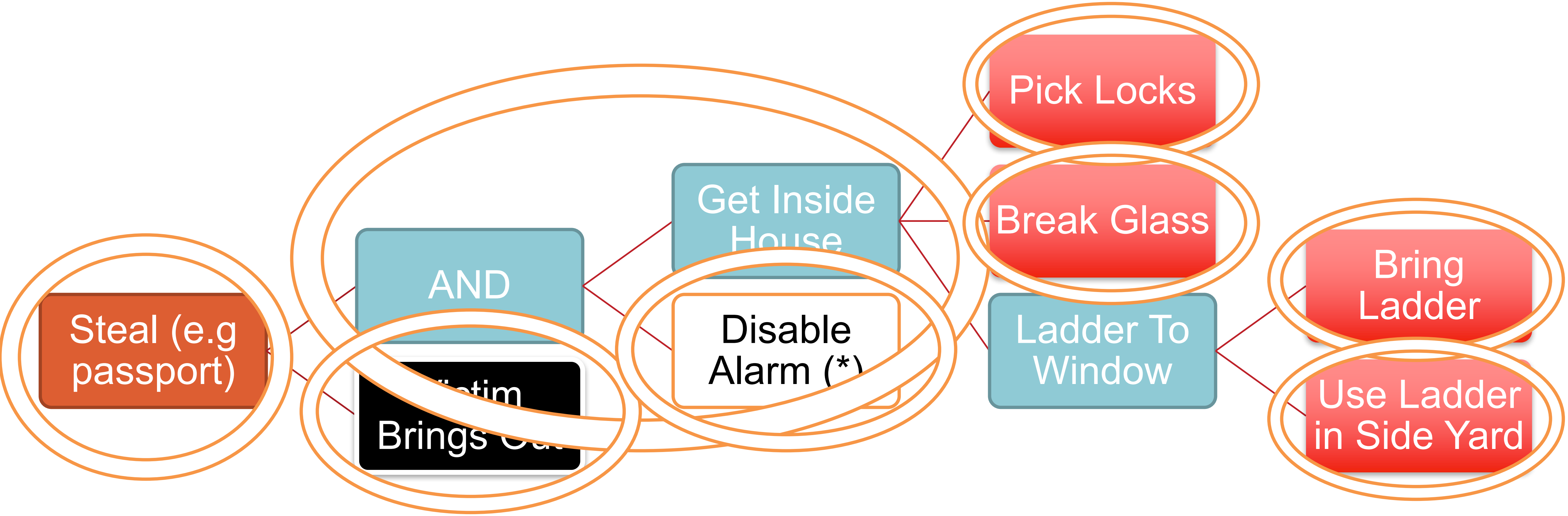
- A model of…
- the many ways to achieve a (nefarious) goal





GENIVI®

Anatomy of an Attack Tree



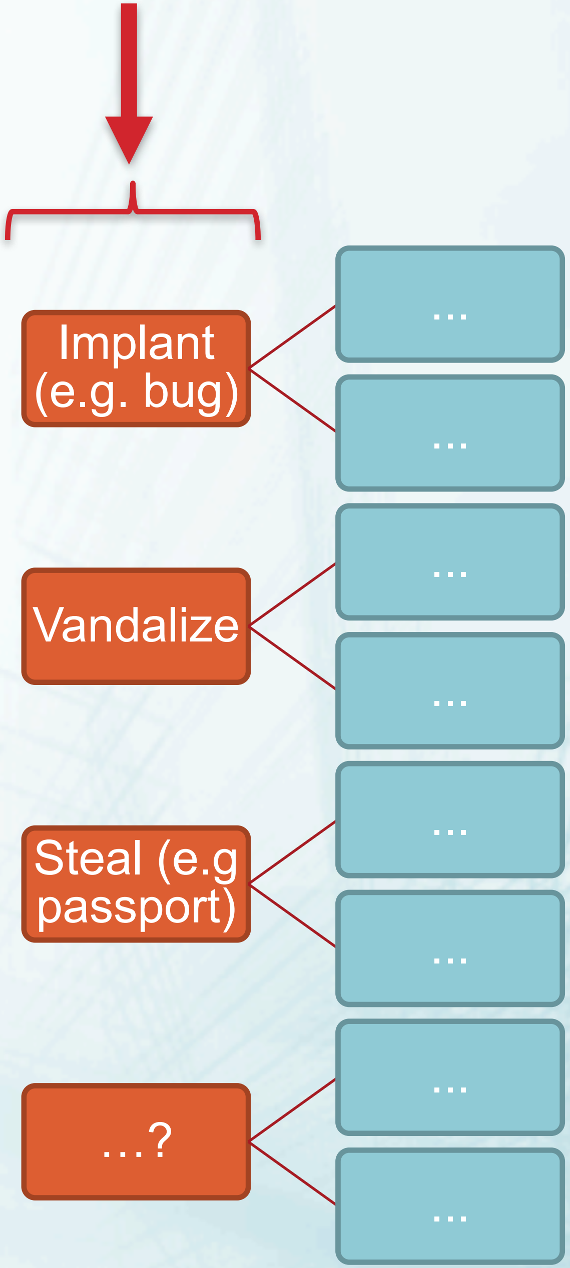
To List Attacker Objectives…

- Making a List: things an Attacker might like to do.
- E.g. “Game Over”s or Attacker Money Makers
- It helps to ask: ”What affects the bottom line?”
  - E.g. Revenue Loss, IP Theft, Brand Damage …

GENIVI®

Attacker Objectives are…

- Attacker Objectives have both:
    1. Clear Attacker Motivations
    2. Clear Impacts (Severities) on the Company/Org./Stakeholders



What Are They Good For?

- They are a Tool With Multiple Applications:

- When Narrowly-Focused:
  - Preparing Offensive Plans (pentesting)
  - Considering Causes of Bugs
  - Brainstorming Defenses
- When Broadly-Focused:
  - **Threat Assessments**

GENIVI®

Threat Assessments

1. …(revealed later)
2. …(revealed later)
3. …(revealed later)
4. A list of mitigations

GENIVI®

You Are the Subject Matter Experts

- Your Domain-Specific knowledge is key
- You know the data flow

GENIVI®

Establish a Common Language

- Create an ***Architecture Summary***

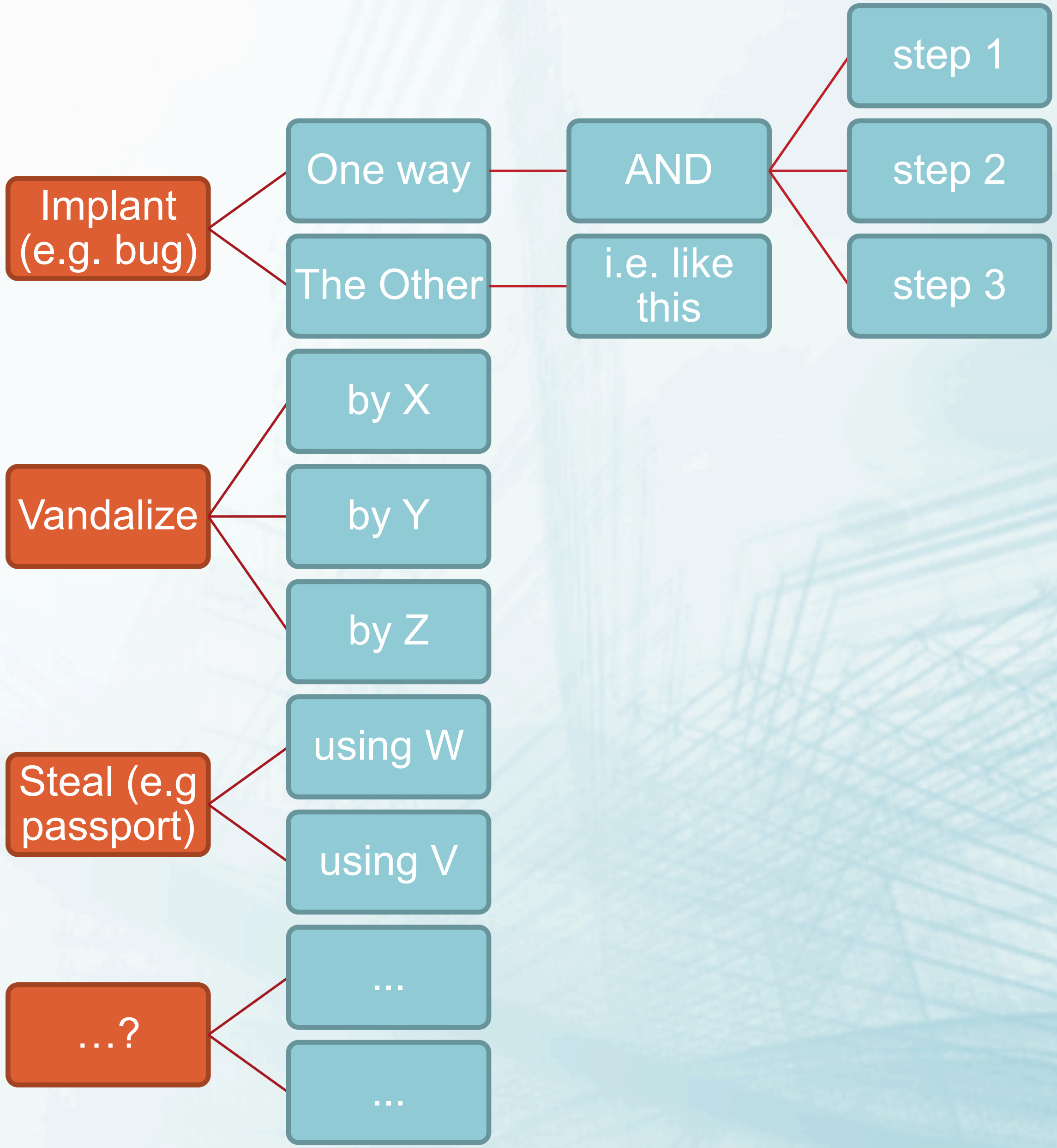
- Do it from your (naïve) perspective

- The result will
  - highlight knowledge gaps and
  - establish a vocabulary for the Threat Assessment

- Capture the data flows in the system

Generating The Trees

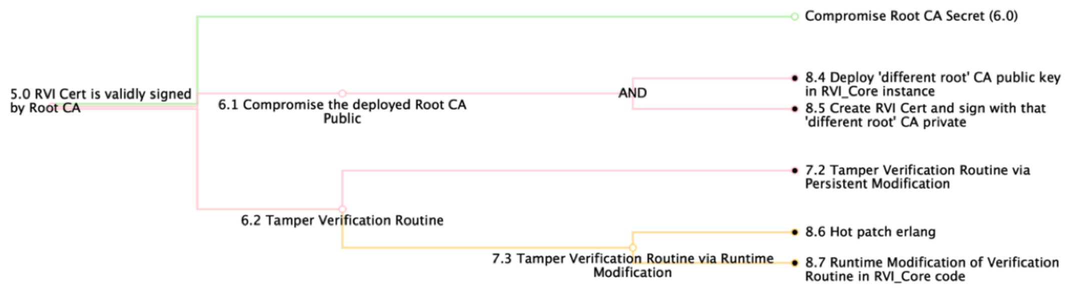
- Do a Tree for Each Asset
- Descend, descend, …
- Worry about *what*, not *how*
  - *Consult your data flows*



Threat Assessments

1. A summary of all attacker objectives
2. A detailed look at the attack vector nodes of the trees
3. An analysis of risk (of the objectives)
4. A list of mitigations

Subtree 5.0 RVI Cert is validly signed by Root CA



5.0 RVI Cert is validly signed by Root CA Attack Subtree

Attack Vector Node 8.4 Deploy 'different root' CA public key in RVI_Core instance

An attacker tampers with a deployed RVI_Core instance's resources where the public key of the root CA is stored. They replace this public key with their own so that they can spoof the authentication server. If they spoof the authentication server then they can generate their own credentials.

Mitigation Required

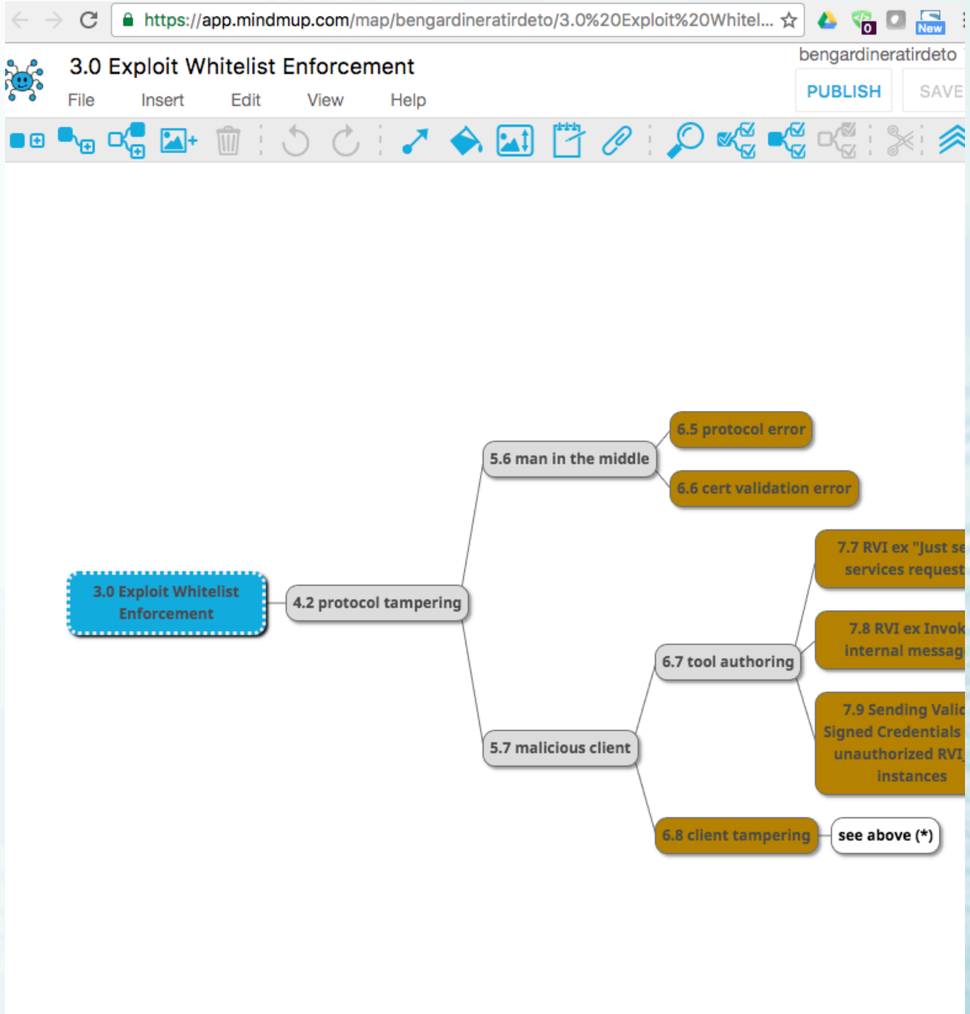
For Implementors of RVI

Credentials for RVI_Core need to be protected a rest and in memory against tampering (and replacement).

GENIVI®

Tools For Attack Trees

- Word Smart-Art
- Visio
- Omnigraffle
- Graphviz DOT
- Any indented text
- Mindmup (e.g. at right)



GENIVI®

See? You CAN do this!

- Threat Assessments up-front will save time
- Threat Assessments up-front will target efforts

- You are the SMEs. You can do this

Your Future:

- Releases with no re-designs
- Threat Assessments in the design phases

GENIVI®

Thank you!

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

Contact us: help@genivi.org

GENIVI®