Stanford Security Seminar

15 March 2016

Words Matter

Language In Security Engineering

Chris Palmer

palmer@google.com

noncombatant.org/security-seminar

About Me

Chrome Security engineer since 2011

- Generalist
- Currently focusing on secure usability

Formerly

- Security engineering consultant with iSEC Partners
- Technology Director at Electronic Frontier Foundation
- Web developer
- Linguist :)

Computers Are Languages

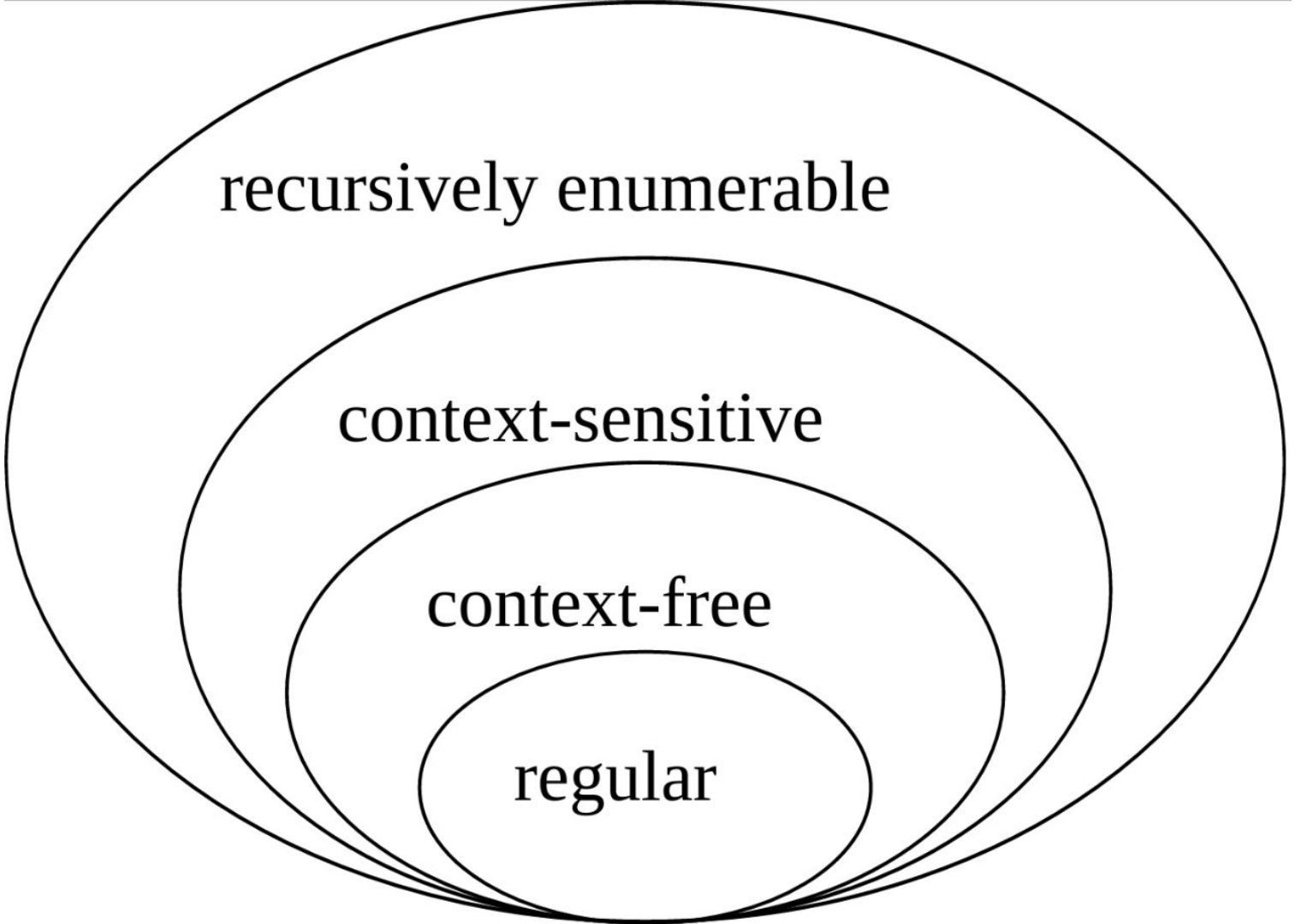
More Languages, More Problems

A browser's job is to parse and interpret utterances, provided by the world's least trustworthy sources (the internet), from many grammars:

- JavaScript, JSON
- PDF
- PNG, JPEG, GIF
- SVG
- Fonts
- Video
- HTML, XML, XSLT
- URLs are not exactly simple, either...

A Linguist’s View Of Software

- A **grammar** describes the set of all and only the **utterances** that make up a **language**
- A **machine** is a **recognizer** for a grammar (input)
- A machine is a **generator** for a grammar (output)
- Particular grammars are members of **equivalence classes**
- Grammar classes form a hierarchy of subsets and supersets
- Programs are utterances, generated and recognizable by machines
- Data (input and output) are utterances, generated and recognizable by machines
- **Programs (and data) are utterances that are also machines**
- Hardware is simply ‘frozen’ software


recursively enumerable
context-sensitive
context-free
regular

Ever tried to parse HTML with regular expressions?

Grammar Class	Machine Type
Regular	Finite state automaton (NFA, DFA)
Context-free	Non-deterministic finite-state automaton
Context-sensitive	Linear bounded automaton
Recursive	Always-halting Turing machine
Recursively enumerable	Turing machine

Language-Theoretic Security

The paranoid linguist's view of software.

#langsec is the insight that **data are utterances that may also be programs that may also be weird machines.**

Weird machines are machines that recognize/generate languages on a surprisingly and/or terrifyingly high position on the Chomsky hierarchy.

The Page-Fault Weird Machine (Bangert, Bratus, Shapiro, Smith)

"Any sufficiently complex input is indistinguishable from byte code" for a weird VM.

Language-Theoretic Security

Another example: the `printf` “little language” is a weird machine, thanks to `%n`.

Format String Attacks (Newsham)

This machine (`main`) recognizes machines (arbitrary `argv[1]`s) that will do whatever they want to your machine (MacBook Pro):

```
int main(int argc, char **argv) {
  char buf[100];
  int x;

  if(argc != 2)
    exit(1);

  x = 1;

  snprintf(buf, sizeof buf, argv[1]);
  buf[sizeof buf - 1] = 0;
  printf("buffer (%zu): %s\n", strlen(buf), buf);
  printf("x is %d/%#x (@ %p)\n", x, x, &x);
}
```

Postel Was Wrong (Thomson)

“In general, an implementation should be conservative in its sending behavior, and liberal in its receiving behavior.” — Jon Postel, trying desperately to get the planet-wide language machine to just to *boot*, let alone actually send anybody’s utterances.

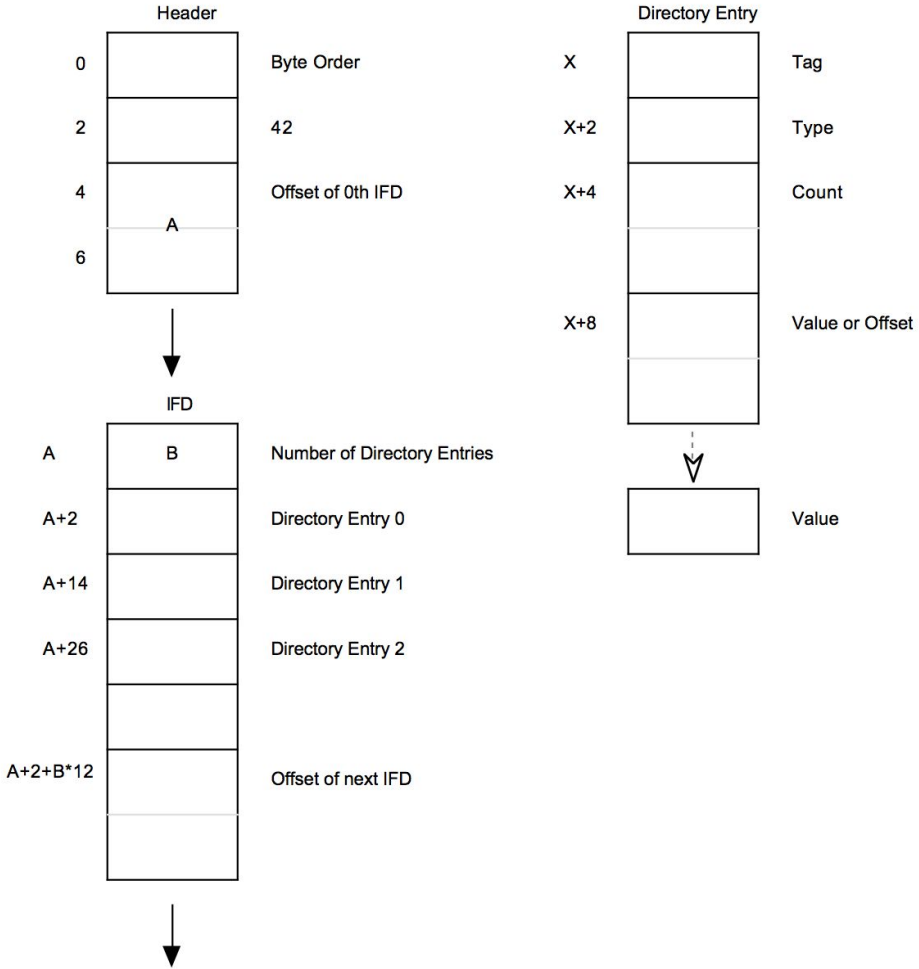
Increasingly, we are turning away from Postel’s Law — out of necessity. Your compiler should detect and reject untrustworthy format specifiers, and your libc shouldn’t implement %n.

```
chris@goatbeast:~ $ make printf-weird
clang -Wall -Wextra -Werror printf-weird.c -o printf-weird
printf-weird.c:12:29: error: format string is not a string
literal (potentially insecure) [-Werror,-Wformat-security]
  snprintf(buf, sizeof buf, argv[1]);
                            ^~~~~~
```

Input Complexity: TIFF

The TIFF image file format specification describes the file format as serialized `structs`. Here is part of the ‘baseline’ TIFF definition; see if you can spot the dormant cyber pathogen:

Figure 1


Header
0
Byte Order
2
42
4
Offset of 0th IFD
A
6
IFD
A
B
Number of Directory Entries
A+2
Directory Entry 0
A+14
Directory Entry 1
A+26
Directory Entry 2
A+2+B*12
Offset of next IFD
Directory Entry
X
Tag
X+2
Type
X+4
Count
X+8
Value or Offset
Value

Input Complexity: H.264

“A specification that cannot be fit on one 8.5 x 11 inch piece of paper cannot be understood.” (Mark Ardis)

The H.264 specification is 263 pages. And not of `struct` definitions or even a BNF grammar, but pseudo-C code. Here is a simple example:

nal_unit(NumBytesInNALunit) {	C	Descriptor
forbidden_zero_bit	All	f(1)
nal_ref_idc	All	u(2)
nal_unit_type	All	u(5)
NumBytesInRBSP = 0		
for(i = 1; i < NumBytesInNALunit; i++) {		
if(i + 2 < NumBytesInNALunit && next_bits(24) == 0x000003) {		
rbsp_byte[NumBytesInRBSP++ **]**	All	b(8)
rbsp_byte[NumBytesInRBSP++ **]**	All	b(8)
i += 2		
emulation_prevention_three_byte /* equal to 0x03 */	All	f(8)
} else		
rbsp_byte[NumBytesInRBSP++ **]**	All	b(8)
}		
}		

Challenges

So, there’s no hope of migrating to a langsec-approved suite of languages (context-free and regular) anytime soon.

Nor is C++ safe.

So we have an unsafe language, parsing complex inputs, at high speed.

My kind of party!

Coping Strategies

We use all the clang/LLVM sanitizers:

- Address Sanitizer
- Undefined Behavior Sanitizer
- Thread Sanitizer

… on a large cluster of machines running fuzzers 24/7

We also use our own `base/numerics` for safe integer arithmetic. (Everybody needs one. Ours is complete, performant, and open source!)

Performance vs. Security

We can’t use `dynamic_cast`: RTTI bloats the code, and Chrome is already plenty big.

So how can we safely cast objects with virtual methods?

Static or debug-only checks for dynamic types don’t work. Let’s take a look...

branches/chromium/1132/Source/WebCore/dom/EventDispatcher.cpp

r110314	r118879	
67	67	`    // as a deeply cloned child of the 'use' element, except that events are dispatched to the SVGElementInstance objects`
68	68	`    Element* shadowHostElement = referenceNode->treeScope()->rootNode()->shadowHost();`
69		`    // At this time, SVG nodes are not allowed in non-<use> shadow trees, so any shadow root we do`
70		`    // have should be a use. The assert and following test is here to catch future shadow DOM changes`
71		`    // that do enable SVG in a shadow tree.`
72		`    ASSERT(!shadowHostElement || shadowHostElement->hasTagName(SVGNames::useTag));`
73		`    if (shadowHostElement && shadowHostElement->hasTagName(SVGNames::useTag)) {`
74		`        SVGUseElement* useElement = static_cast<SVGUseElement*>(shadowHostElement);`
75		
76		`        if (SVGElementInstance* instance = useElement->instanceForShadowTreeElement(referenceNode))`
77		`            return instance;`
78		`    }`
	69	`    // At this time, SVG nodes are not supported in non-<use> shadow trees.`
	70	`    if (!shadowHostElement || !shadowHostElement->hasTagName(SVGNames::useTag))`
	71	`        return referenceNode;`
	72	`    SVGUseElement* useElement = static_cast<SVGUseElement*>(shadowHostElement);`
	73	`    if (SVGElementInstance* instance = useElement->instanceForShadowTreeElement(referenceNode))`
	74	`        return instance;`
79	75	`#endif`
80	76	

branches/chromium/1132/Source/WebCore/page/EventHandler.cpp

r115877	r118879	
2097	2097	
2098	2098	`    Element* shadowTreeParentElement = shadowTreeElement->shadowHost();`
2099		`    if (!shadowTreeParentElement)`
	2099	`    if (!shadowTreeParentElement || !shadowTreeParentElement->hasTagName(useTag))`
2100	2100	`        return 0;`
2101	2101	
2102		`    ASSERT(shadowTreeParentElement->hasTagName(useTag));`
2103	2102	`    return static_cast<SVGUseElement*>(shadowTreeParentElement)->instanceForShadowTreeElement(referenceNode);`
2104	2103	`}`

A Thought On Types

wrds_mttr_t

Even in 2016, C programmers believe the word `int` refers to the mathematical concept *integer*. Even programmers who should know better, such as kernel programmers. It's not even (necessarily) defined as modular arithmetic (!).

`grep -ri alloc *` in your favorite C/C++ codebase.

The C and C++ language specifications leave the behavior of signed `int`s undefined! (Fun game: Why?)

The words are simply incapable of carrying the concepts we load them with.

Integers are just the beginning...

Typeful Programming (Cardelli)

“…there must be a mechanical way of verifying that type constraints are respected by programs. … *laws should be enforceable*: unchecked constraining information, while often useful for documentation purposes, cannot be relied upon and is very hard to keep consistent in large software systems. In general, systems should not exhibit constraints that are not actively enforced at the earliest possible moment. In the case of typechecking the earliest moment is at compile-time, although some checks may have to be deferred until run-time. In contrast, some specifications can be neither typechecked nor deferred until run time, and require general theorem-proving (e.g., in verifying the property of being a constant function).

“Another emphasis is on transparent typing. It should be easy for a programmer to predict reliably which programs are going to typecheck.”

Typeful Programming

Cardelli is concerned with **reliability**, by which he means (part of) what I call security.

The paper is a pragmatic, readable approach to the concept of **propositions as types** (Wadler); propositions include security guarantees.

The paper and the language it describes, Quest, provide a framework for understanding what we can (and should) expect from the type systems in our real-world programming languages.

An Example: Origins Are Not URLs

Chrome historically represented **the web origin concept** (Barth) with a class called `GURL` — overloading the concept of URLs to include origins.

But an origin is not just the (scheme, host, port) sub-tuple of a URL tuple.

- `data:`, `blob:`, `filesystem:`, `javascript:`, `chrome:`, `chrome-extension:` …
- `null`, “”
- `file:` URLs can include hostnames; but the pathname is the distinguisher
- The predicates don’t quite line up: `empty`, `valid`, `equals`

An Example: Origins Are Not URLs

Since the origin is the fundamental boundary/isolation mechanism/principal on the web, it's Kind Of A Big Deal to get the concept right.

Mike West introduced a new `url::Origin` class, but we are still trying to retrofit the new concept to old code.

But there are other ways in which typeful programming could have helped us avoid problems...

```
void SavePackage::OnReceivedSavableResourceLinksForCurrentPage(
    const std::vector<GURL>& resources_list,
    const std::vector<Referrer>& referrers_list,
    const std::vector<GURL>& frames_list) {
  if (wait_state_ != RESOURCES_LIST)
    return;

  DCHECK(resources_list.size() == referrers_list.size());
...
  if (all_save_items_count_) {
    // Put all sub-resources to wait list.
    for (int i = 0; i < static_cast<int>(resources_list.size()); ++i) {
      const GURL& u = resources_list[i];
      DCHECK(u.is_valid());
      SaveFileCreateInfo::SaveFileSource save_source = u.SchemeIsFile() ?
          SaveFileCreateInfo::SAVE_FILE_FROM_FILE :
          SaveFileCreateInfo::SAVE_FILE_FROM_NET;
      SaveItem* save_item = new SaveItem(u, referrers_list[i],
                                         this, save_source);
```

Say What You Mean With Types

One approach to solve that would be to encapsulate the consistency check in the constructor of a new type, or use a `std::map`.

```
class SavableResourceLinks { ... }
```

```
typedef std::map<GURL, Referrer> ResourcesAndReferrers;
```

Why not? Insufficiently expressive/convenient (de)serialization API for IPC? Programmer didn’t think of it?

Protocols

What Is A Protocol?

protocol : computer language :: pragmatics : natural language

In other words, when the *semantics* (as opposed to the syntax) of utterances are context-sensitive.

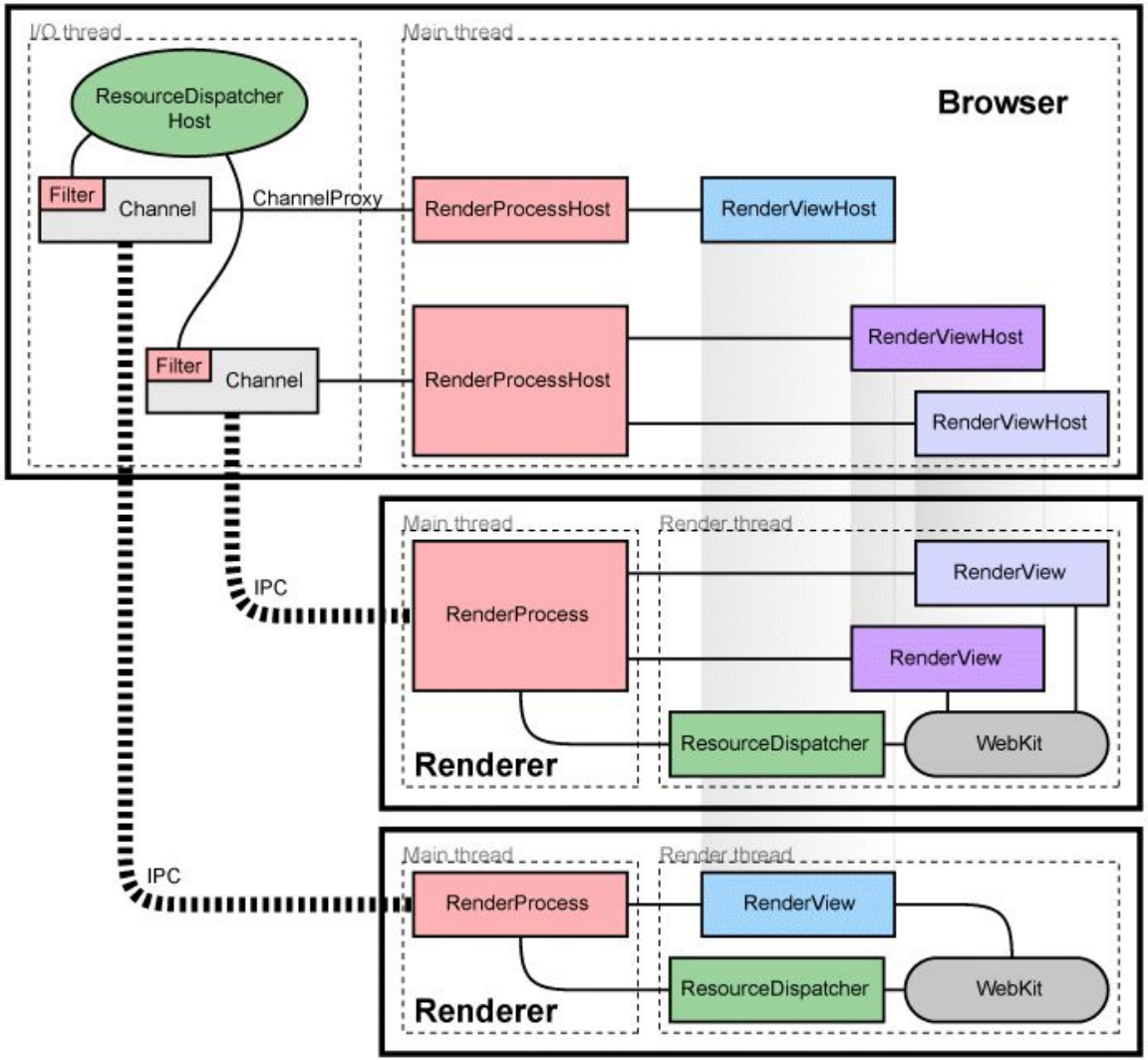
For example: If Alice says `SYN/ACK` to Bob, Bob can't understand it unless Bob (a) said `SYN` to Alice; (b) immediately before.

Interpreting Languages In Isolated Contexts

To change the interpretation context is to change — potentially to reduce — the meaning of what is said. Privilege reduction, privilege separation.

It’s a great idea, and it works.

But, as we saw in the previous section, it trades one language problem (our JavaScript machine might turn out to be a weird machine and allow web content to take over a renderer) for another set of problems: IPC and (painfully optional) run-time type checking!


I/O thread
Main thread
Browser
ResourceDispatcher Host
Filter
Channel
ChannelProxy
RenderProcessHost
RenderViewHost
Filter
Channel
RenderProcessHost
RenderViewHost
RenderViewHost
IPC
Main thread
Render thread
RenderProcess
RenderView
RenderView
ResourceDispatcher
WebKit
Renderer
IPC
Main thread
Render thread
RenderProcess
RenderView
ResourceDispatcher
WebKit
Renderer

Handling Protocols Is Run-Time Type Checking

As Cardelli wisely advises, it is strictly less awesome than static type checking.

As the `static_cast` example showed, it's not always sufficiently performant.

As the `OnReceivedSavableResourceLinksForCurrentPage` example showed, it's not always as convenient or as obvious as it should be.

Ceremonies

Ceremony Design And Analysis (Ellison)

“The concept of **ceremony** is introduced as an extension of the concept of **network protocol**, with human nodes alongside computer nodes and with communication links that include UI, human-to-human communication and transfers of physical objects that carry data. What is out-of-band to a protocol is in-band to a ceremony, and therefore subject to design and analysis using variants of the same mature techniques used for the design and analysis of protocols. Ceremonies include all protocols, as well as all applications with a user interface, all workflow and all provisioning scenarios. A secure ceremony is secure against both normal attacks and social engineering. However, some secure protocols imply ceremonies that cannot be made secure.”

Ceremonies And Unwi[tl]{2}ing Participants

If you thought implementing protocol nodes was hard, due to weak run-time typing, you'll *love* the challenge of serializing layer 4 data structures and deserializing them at layer 8!

Imagine an RPC endpoint that

- Implements a randomly-incomplete deserializer
  - I.e. people don't read all the words we write
- Lacks implementations of some dependencies
  - I.e. people don't always know what our words mean

Your connection is not private

Attackers might be trying to steal your information from **pinningtest.appspot.com** (for example, passwords, messages, or credit cards).

NET::ERR_SSL_PINNED_KEY_NOT_IN_CERT_CHAIN

☐ Automatically report details of possible security incidents to Google. Privacy policy

Advanced

Reload

Poor Pragmatics

Even if the person reading that understands the RSA cryptosystem, there are 2 problems:

- They may lack crucial context (that Chrome has baked-in knowledge of the server's public key)
- This conversation interrupts the one the person thought they were having
    - That is, it disrupts their task and subverts their purpose in using Chrome in the first place

And yet Chrome 'knows' that an attack is underway.

The Tower Of Techno-Babel

It’s hard enough if everyone used English. But, they don’t.

- Polysemy in the source language
  - Cookie store: a place to buy a tasty treat? A database of short strings? A place to buy <weird techno-English word>? [Spanish]
  - Private browsing: Being discrete and maintaining confidentiality? Not taxpayer-funded? [Bulgarian]
- Very different grammar in the target language
  - ‘Microsoft Turkish’
  - ‘Engrish’
  - 怪しい日本語 (“suspicious Japanese”)

Why Are Translation And Internationalization Hard?

- Specialized vocabulary, non-specialist translators
- Context-dependent vocabulary; translators just get a list of isolated phrases and divvy them up (no context during translation)
- We don’t even have a consistent or well-defined vocabulary in the source language (English)!
  - Key, certificate, pin, identity
  - ‘Privacy’, ‘security’
  - Incognito/Off The Record, Private Browsing
  - Site, origin, page, app
- Different cultural associations

We Don’t Even Know What ‘Names’ Are

Falsehoods Programmers Believe About Names (McKenzie)

1. People have exactly one canonical full name.
2. People have exactly one full name which they go by.
3. People have, at this point in time, exactly one canonical full name.
4. People have, at this point in time, one full name which they go by.
5. People have exactly N names, for any value of N.
6. People’s names fit within a certain defined amount of space.
7. People’s names do not change.
8. People’s names change, but only at a certain enumerated set of events.
9. People’s names are written in ASCII.
10. People’s names are written in any single character set.
11. People’s names are all mapped in Unicode code points.
12. People’s names are case sensitive.
13. People’s names are case insensitive.

It goes on...

Visual Language Is Hard, Too

A language is a grammar of symbols — but words are not the only kinds of symbols.

- Spoken words
- Written words
- Sign language
- Body language
- Color, iconography, shape
- Pliancy, flat design, skeumorphism, cultural associations

Visual Language Is Crucial, And Complicated

Visual language is powerful, and often overrides words.


kate spade

“Oh, it’s telling me I’m on a shopping site.”

You've gone incognito

Pages you view in incognito tabs won't stick around in your browser's history, cookie store, or search history after you've closed all of your incognito tabs. Any files you download or bookmarks you create will be kept.

However, you aren't invisible. Going incognito doesn't hide your browsing from your employer, your internet service provider, or the websites you visit.

LEARN MORE

“If it’s good enough for Edward Snowden, it’s good enough for me!”

Conclusion

Security Engineering Is Adversarial Linguistics

- Mean what you say
  - Say what you mean
- Be able to express what you mean in whatever language is available
  - Without making surprisingly powerful languages available to your adversary
- Prove that what you say is true
  - Prove that what they say is true
- Be aware of whom you are talking to
  - And who is talking to you

We have to talk to every machine, and to every human, truthfully.

Happy Parsing!