ROMA MMXVI | APPSEC EUROPE

Attack Patterns for Black-Box Detection of Logical Vulnerabilities in Multi-Party Web Applications

Avinash Sudhodanan and Luca Compagna

j.w.w.

Alessandro Armando, Roberto Carbone

Adrien Hubner, Nicolas Dolgin

About Us

Avinash Sudhodanan

- Final year PhD student at U. of Trento
- Early Stage Researcher at FBK (SECENTIS)

Alessandro Armando

- Head of Research Unit (Security & Trust, FBK)
- Associate Professor (U. of Genova)

Roberto Carbone

- Researcher (Security & Trust, FBK)

Luca Compagna

- Researcher (SAP)

Adrien Hubner

- Intern (SAP)

Nicolas Dolgin

- Intern (SAP)

Agenda

Introduction & Problem

- Multi-Party Web Applications (MPWAs)
- Logical Vulnerabilities in MPWAs
- Detecting Attacks caused by Logical Vulnerabilities

Observations & Solution

- Attacks to Attack Pattern
- Attack Pattern-based Security Testing

Results & Demo

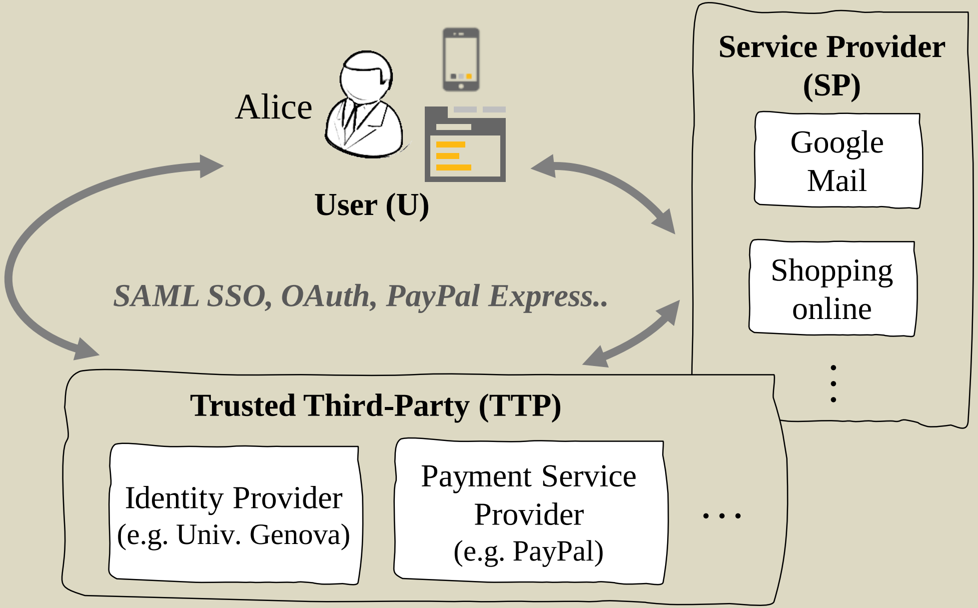
Industrial Exploitation, Limitations & Future Work

Agenda

- Introduction & Problem
  - Multi-Party Web Applications (MPWAs)
  - Logical Vulnerabilities in MPWAs
  - Detecting Attacks caused by Logical Vulnerabilities
- Observations & Solution
  - Attacks to Attack Pattern
  - Attack Pattern-based Security Testing
- Results & Demo
- Industrial Exploitation, Limitations & Future Work

Multi-Party Web Applications (MPWAs)

A **Service Provider** web app. relying on **Trusted Third-Parties** to deliver its services to **Users**

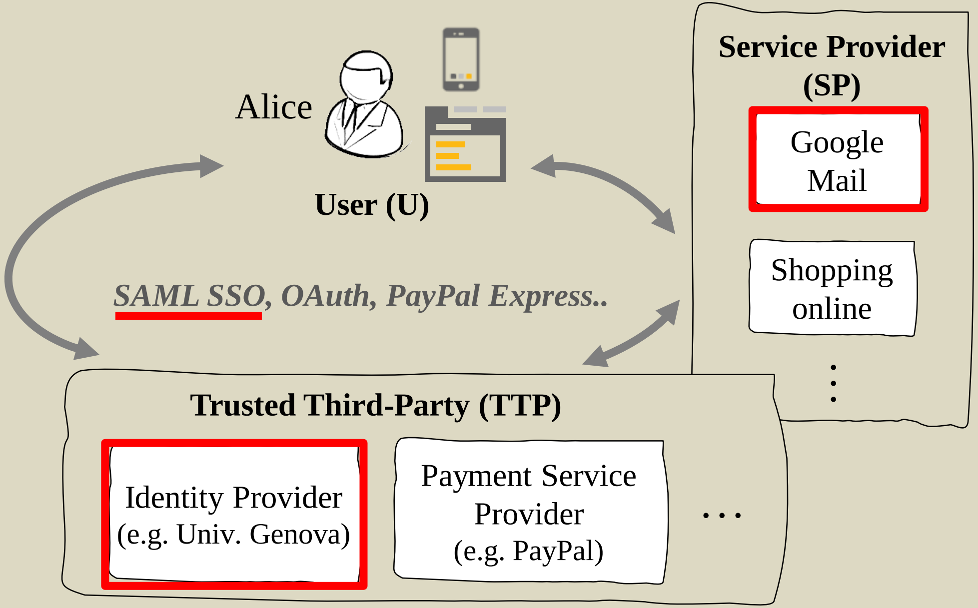


Multi-Party Web Applications (MPWAs)

A **Service Provider** web app. relying on **Trusted Third-Parties** to deliver its services to **Users**

Examples

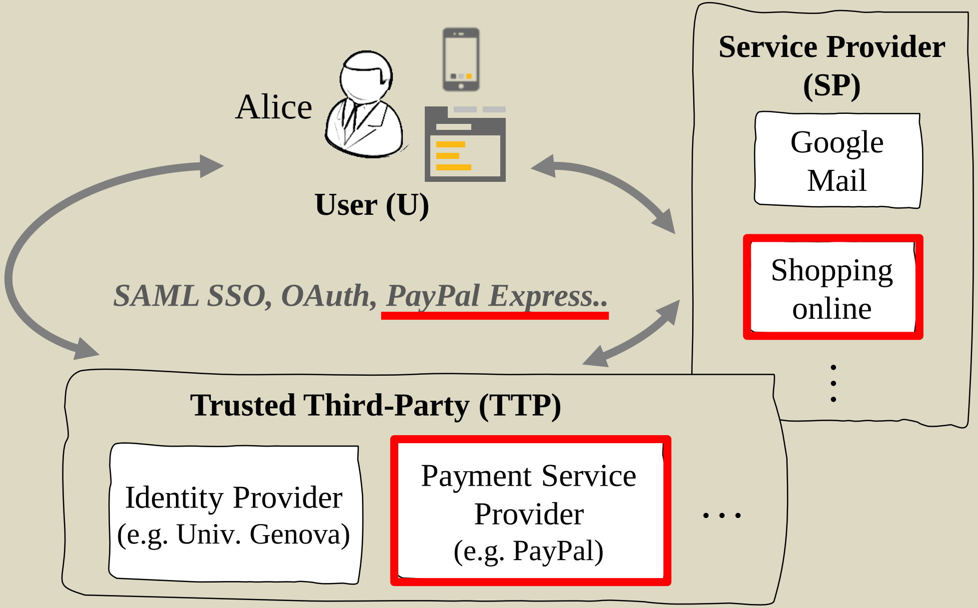
- Single Sign-On (SSO)



Multi-Party Web Applications (MPWAs)

A **Service Provider** web app. relying on **Trusted Third-Parties** to deliver its services to **Users**

Examples
- Single Sign-On (SSO)
- Cashier-as-a-Service (CaaS)

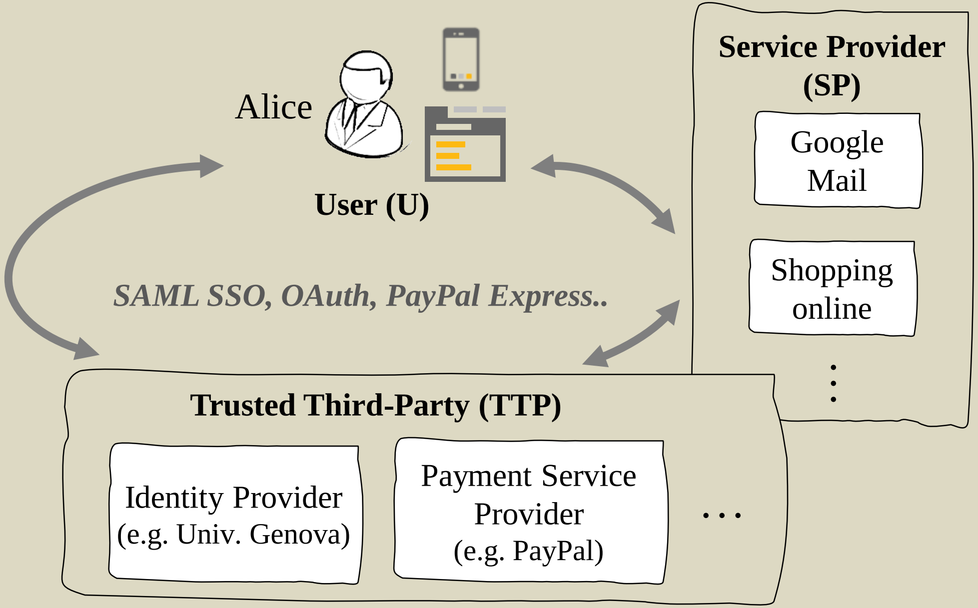


Multi-Party Web Applications (MPWAs)

A **Service Provider** web app. relying on **Trusted Third-Parties** to deliver its services to **Users**

Examples
- Single Sign-On (SSO)
- Cashier-as-a-Service (CaaS)

Popularity/Relevance
- 27% of top 1000 US websites supports Facebook SSO [USENIX'14]
- 180+ million PayPal users worldwide

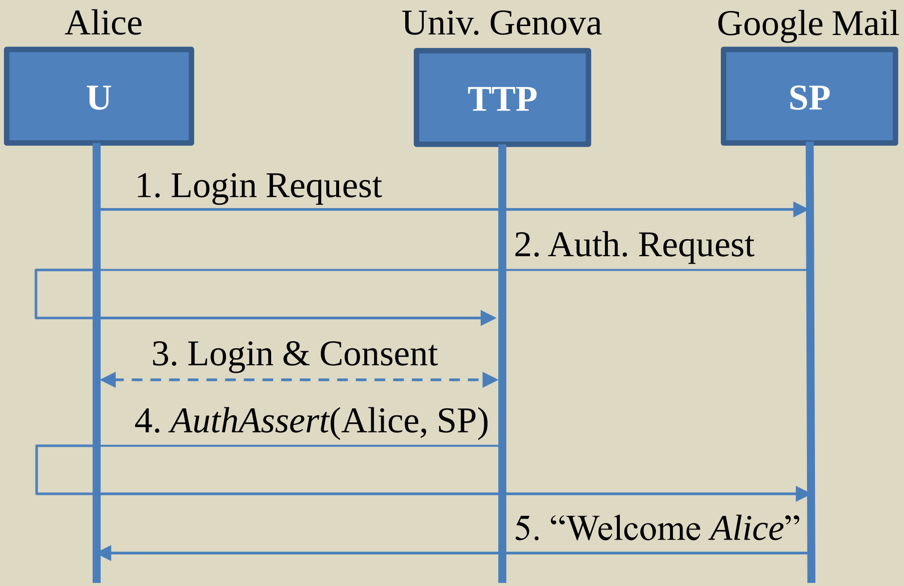


Multi-Party Web Applications (MPWAs)

A **Service Provider** web app. relying on **Trusted Third-Parties** to deliver its services to **Users** (via **web-based security protocols**)

Examples
- Single Sign-On (SSO)
- Cashier-as-a-Service (CaaS)

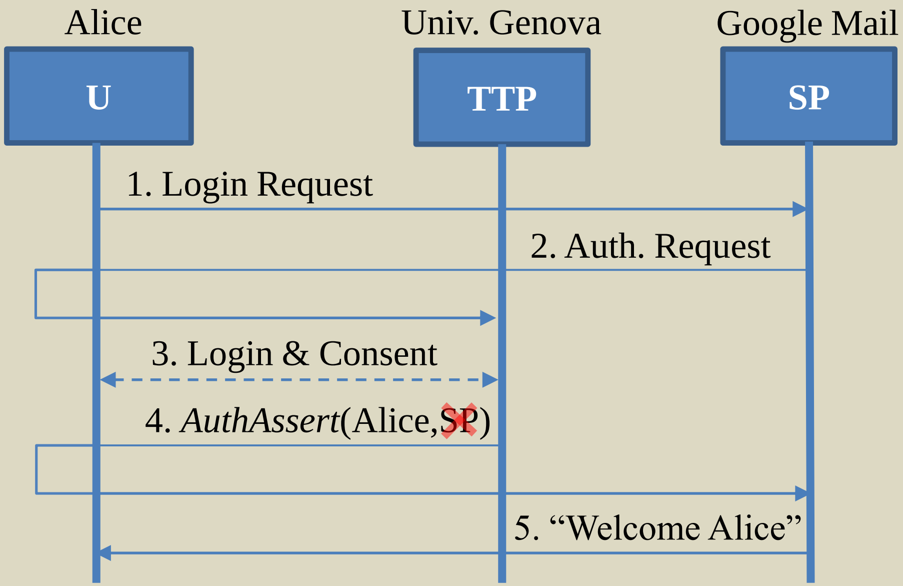
Popularity/Relevance
- 27% of top 1000 US websites supports Facebook SSO [USENIX’14]
- 180+ million PayPal users worldwide



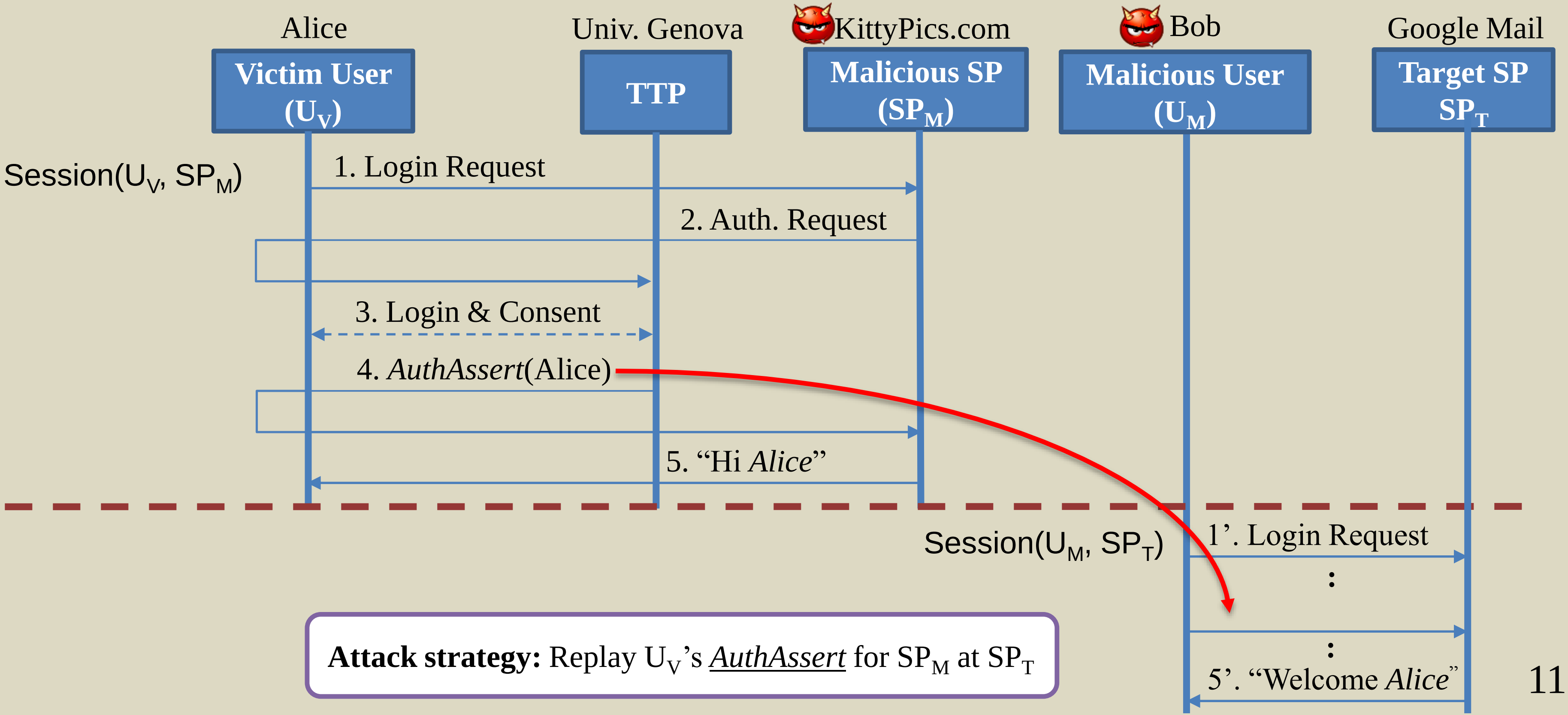
Logical Vulnerabilities in MPWAs

Caused by incorrect logic of the **design/implementation** of the protocols underlying MPWAs (e.g., [FMSE ‘08, NDSS ‘13, USENIX ’13])

Example: Vulnerability in SAML-based SSO for Google Apps was reported [FMSE ‘08]



Attack on SAML-based SSO for Google Apps



Attack strategy: Replay U_V's *AuthAssert* for SP_M at SP_T

Detecting Attacks Exploiting Logical Vulns.

Attacks reported in the past were discovered using a **variety of techniques** applied to **specific scenarios**

Tech. [Ref.]	Vulnerable MPWA	Attack Strategy	Attacker's Goal
Formal Verification [FMSE'08]	SAML SSO for Google Apps (SPs)	Owner of a malicious SP (SP_M) replays victim user's (U_V's) *AuthAssert* for SP_M at target SP (SP_T)	Authenticate as U_V at SP_T
Grey-Box+Formal Verification [NDSS'13]	*developer.mozilla.com* (SP) implementing BrowserID	Malicious user (U_M) sends his/her *AuthAssert* for SP_T through U_V's browser	Authenticate U_V as U_M at SP_T
Black-Box [NDSS'14]	PayPal Express Checkout in OpenCart 1.5.3.1	Malicious user (U_M) replays *Token* of a completed purchase during a new purchase at SP_T	Successfully complete new purchase at SP_T
Formal Verification [USENIX'13]	SPs implementing Facebook SSO	Owner of a malicious SP (SP_M) replays victim user's (U_V's) *AccessToken* for SP_M at target SP (SP_T)	Authenticate as U_V at SP_T
White-Box [NDSS'14]	*Authorize.net* credit card sim in baby products store	Malicious user (U_M) replays *OrderId* of a completed purchase during a new purchase at SP_T	Successfully complete new purchase at SP_T
Formal Verification [CSF'11]	*CitySearch.com* (SP) using Facebook SSO	Malicious user (U_M) sends his/her *AuthCode* for SP_T through U_V's browser	Authenticate U_V as U_M at SP_T

*Can we elaborate a **viable, scenario-agnostic technique** to detect all these kind of attacks?*

Agenda

Introduction & Problem

- Multi-Party Web Applications (MPWAs)
- Logical Vulnerabilities in MPWAs
- Detecting Attacks caused by Logical Vulnerabilities

Observations & Solution

- Attacks to Attack Pattern
- Attack Pattern-based Security Testing

Results & Demo

Industrial Exploitation, Limitations & Future Work

Our Observation- I: Attack Strategies

The strategy behind many attacks reported in the literature is the same

Tech. [Ref.]	Vulnerable MPWA	Attack Strategy (simplified)	Attacker's Goal
Formal Verification [2]	SAML SSO for Google Apps (SPs)	REPLAY *AuthAssert* from Session(U_V,SP_M) in Session(U_M, SP_T)	Authenticate as U_V at SP_T
Grey-Box+Formal Verification [3]	*developer.mozilla.com* (SP) implementing BrowserID	SEND REQUEST-OF *AuthAssert* FROM Session(U_M, SP_T) through U_V's browser	Authenticate U_V as U_M at SP_T
Black-Box [4]	PayPal Express Checkout in OpenCart 1.5.3.1	REPLAY *Token* from Session(U_M,SP_T) in Session(U_M,SP_T)	Successfully complete new purchase at SP_T
Formal Verification [5]	SPs implementing Facebook SSO	REPLAY *AccessToken* from Session(U_V,SP_M) in Session(U_M, SP_T)	Authenticate as U_V at SP_T
White-Box [7]	*Authorize.net* credit card sim in baby products store	REPLAY *OrderId* from Session(U_M,SP_T) in Session(U_M,SP_T)	Successfully complete new purchase at SP_T
Formal Verification [8]	*CitySearch.com* (SP) using Facebook SSO	SEND REQUEST-OF *AuthCode* FROM Session(U_M, SP_T) through U_V's browser	Authenticate U_V as U_M at SP_T

⋮

Can we exploit the similarity in attack strategies to discover new attacks in an automatic way?

Our Observation- II: Sec.-critical Elements

Some properties of the HTTP elements of protocols can be used as **preconditions** to apply the attack strategy:

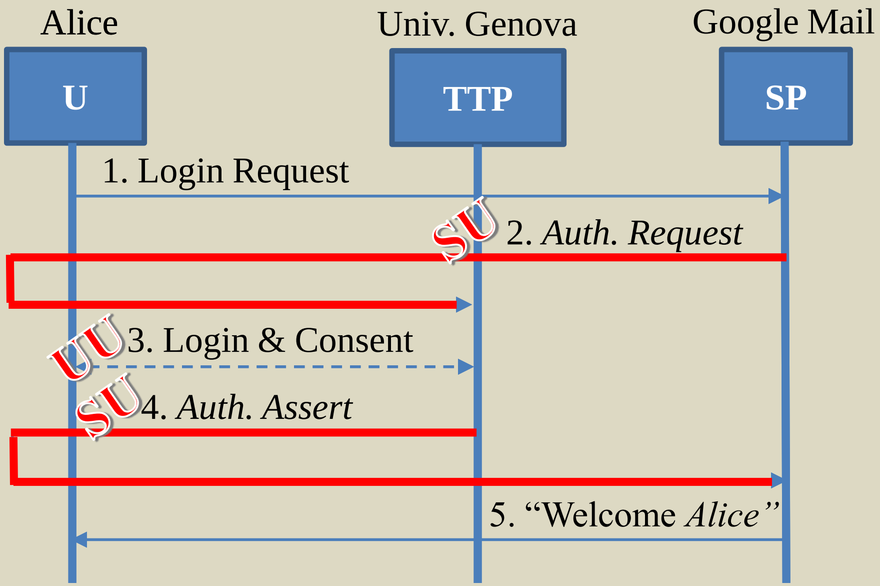
- **Syntactic/Semantic properties** of HTTP elements [6]

Property	Label
User Unique	UU
Session Unique	SU

:

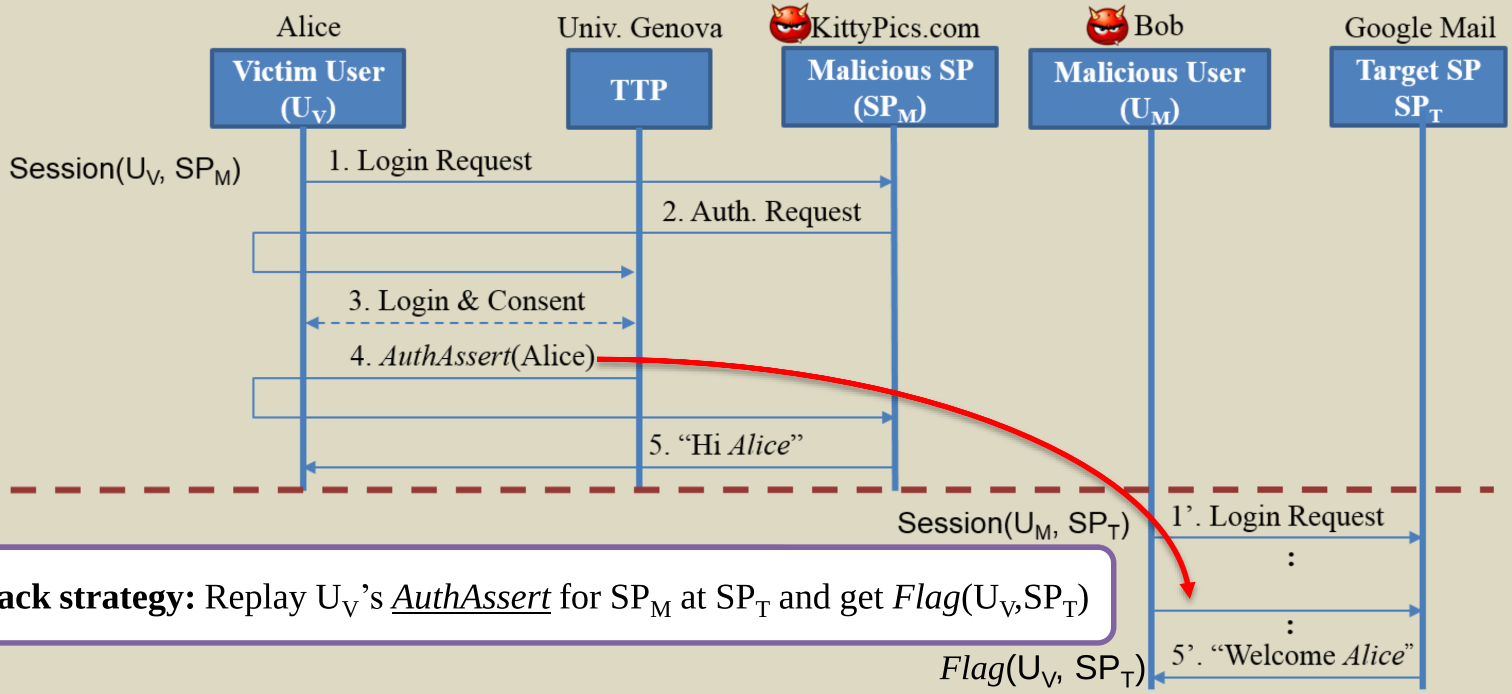
- **Dataflow properties**

Property	Flow
The HTTP element flows from SP to TTP, through the browser	SP-TTP
The HTTP element flows from TTP to SP, through the browser	TTP-SP



We can understand from the HTTP traffic of the underlying protocol which attack strategy to apply!

Observation-III: Postconditions



Attack strategy: Replay U_V's *AuthAssert* for SP_M at SP_T and get *Flag*(U_V,SP_T)

We can determine the successful execution of an attack strategy through observable DOM/traffic patterns!

Our Observation- IV: Threat Model

Attacker can play the role of a **User** and/or a **Service Provider**

– Four nominal sessions are sufficient to execute all the attacks we considered:

Nominal Sessions			
#	**User**	**SP**	**Comment**
S_1	U_V	SP_T	Session between potential victim, target SP and TTP
S_2	U_M		Session between malicious user, target SP and TTP
S_3	U_V	SP_M	Session between potential victim, reference SP and TTP
S_4	U_M		Session between malicious user, reference SP and TTP

Is this threat model sufficient? Any added value by considering browser history attacker?

From Attacks to Attack Pattern

Tech. [Ref.]	Vulnerable MPWA	Attack Strategy	Attacker's Goal
Formal Verification [2]	SAML SSO for Google Apps (SPs)	Owner of a malicious SP (SP_M) replays victim user's (U_V's) *AuthAssert* for SP_M at target SP (SP_T)	Authenticate as U_V at SP_T
Formal Verification [5]	SPs implementing Facebook SSO	Owner of a malicious SP (SP_M) replays victim user's (U_V's) *AccessToken* for SP_M at target SP (SP_T)	Authenticate as U_V at SP_T

Tech. [Ref.]	Formalized Attack Strategy
Formal Verification [2]	REPLAY *AuthAssert* FROM Session(U_V, SP_M) IN Session(U_M, SP_T)
Formal Verification [5]	REPLAY *AccessToken* FROM Session(U_V, SP_M) IN Session(U_M, SP_T)

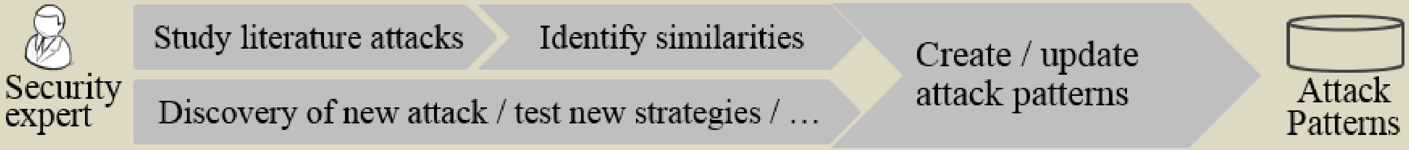
Name	Attack Strategy	Precondition	Postcondition
Type 1 Replay Attack (RA1)	REPLAY *x* FROM Session(U_V, SP_M) IN Session(U_M, SP_T)	TTP-SP ∈ *x*.flow AND SU|UU ∈ *x*.labels	*Flag*(U_V, SP_T) e.g. "Welcome *Alice*"

Attack Patterns

Name	Attack Strategy	Precondition	Postcondition
RA1	REPLAY x FROM (U_V, SP_M) IN (U_M, SP_T)	(TTP-SP $\in x$.flow AND $(SU\|UU) \in x$.labels)	(U_V, SP_T)
RA2	REPLAY x FROM (U_M, SP_M) IN (U_M, SP_T)	(SP-TTP $\in x$.flow AND $(SU\|AU) \in x$.labels)	(U_M, SP_T)
RA3	REPLAY x FROM (U_M, SP_T) IN (U_M, SP_T)	(TTP-SP $\in x$.flow AND SU $\in x$.labels)	(U_M, SP_T)
RA4	REPLAY y FROM S IN (U_M, SP_T) where $S =$ REPLAY x FROM (U_M, SP_T) IN (U_V, SP_M)	(SP-TTP $\in x$.flow AND $(SU\|AU) \in x$.labels AND TTP-SP $\in y$.flow AND $(SU\|UU) \in y$.labels)	(U_V, SP_T)
LCSRF	REPLACE req WITH REQUEST-OF y FROM (U_M, SP_T) IN $[U_M$ SEND $req]$	(TTP-SP $\in y$.flow AND $(SU\|UU) \in y$.labels)	(U_M, SP_T)
RedURI	REPLAY y FROM S IN (U_M, SP_T) where $S =$ REPLACE x WITH x' IN (U_V, SP_T)	(SP-TTP $\in x$.flow AND RURI $\in x$.labels) AND TTP-SP $\in y$.flow AND $(SU\|UU) \in y$.labels)	(U_M, SP_T)
RA5	REPLAY x FROM (U_V, SP_T) IN (U_M, SP_T)	(TTP-SP $\in x$.flow AND $(SU\|UU) \in x$.labels AND x.location = REQUESTURL)	(U_V, SP_T)

Approach

- Knowledge of the security expert is **encapsulated** in **attack patterns**



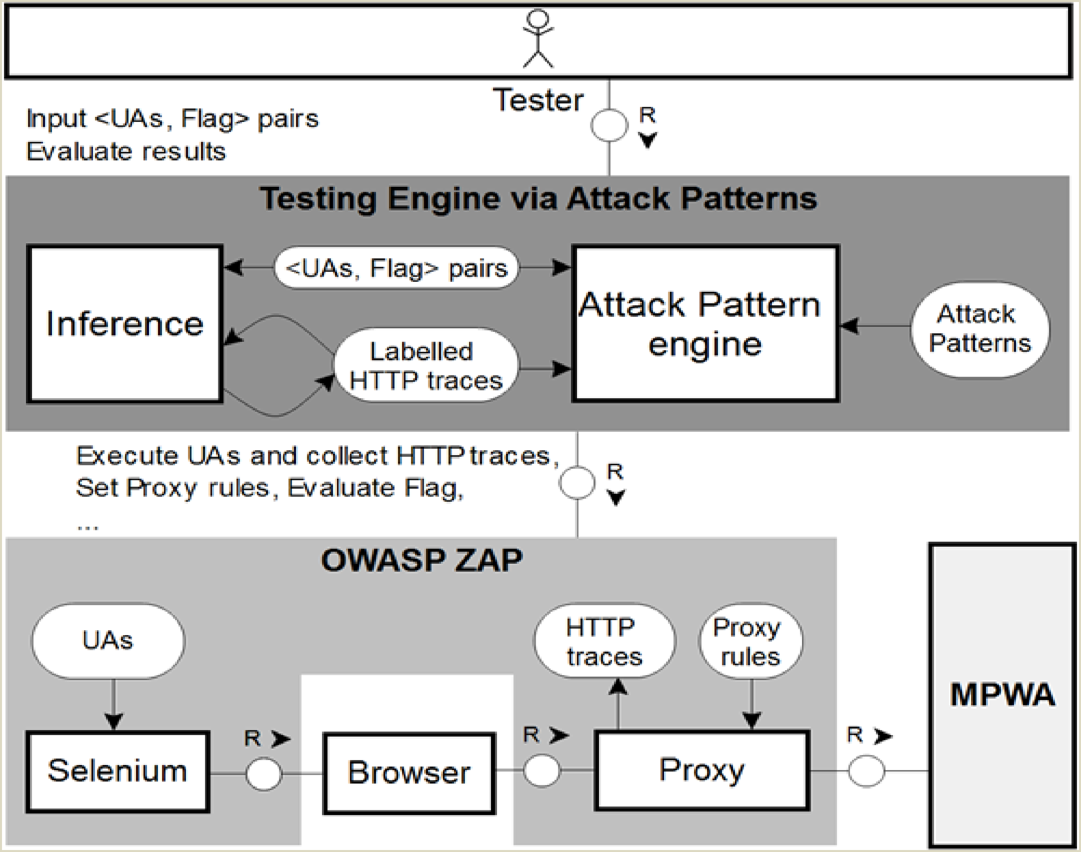
- We provide a framework for the tester of a MPWA to apply the attack patterns to detect attacks

- Provide **implementation**, recording of **user actions** of the **nominal sessions**

- Execute user actions
- Identify **syntactic/semantic, data flow** properties of underling HTTP elements (e.g. SU, TTP-SP etc.)

- Check **preconditions**
- Execute **attack strategy** e.g. **replay** an **element** from one protocol run in another
- Check **postconditions**

Implementation



Agenda

Introduction & Problem

- Multi-Party Web Applications (MPWAs)
- Logical Vulnerabilities in MPWAs
- Detecting Attacks caused by Logical Vulnerabilities

Observations & Solution

- Attacks to Attack Pattern
- Attack Pattern-based Security Testing

Results & Demo

Industrial Exploitation, Limitations & Future Work

Results (excerpt)

Novelty	SP	TTP (& Protocol)	Attack (& Elements)	ACKs
New attack	Alexa e-comm < 10	Linkedin JS API SSO	RA5 (*Uid, Email*)	
	developer.linkedin.com		RA5 (*Mem. Id, Access. Token*)	✓
Attacks **previously reported in SSO** and we found them in other scenarios (**CaaS, reg. via email)**	All SPs	Stripe Checkout	RA4 (*DataKey, Token*)	✓
	open.sap.com	Gmail (reg. via email)	LCSRF *(Act. Link)*	✓
Attack **previously reported** in SSO (or CaaS) protocol and we found in **another** SSO (or CaaS) protocol	INstant	Linkedin JS API SSO	RA1 *(Access_Token)*	✓
	Alexa US top < 1000	Log in with Instagram	LCSRF (*Auth. Code*)	
	pinterest.com	Facebook SSO	RedURI *(red_uri, Auth. Code)*	✓
	All SPs	Log in with PayPal	RedURI *(red_uri, Auth. Code)*	✓
Same attack another app	OpenCart v2.1.0.1	2Checkout	RA3 *(Order_num, Key)*	

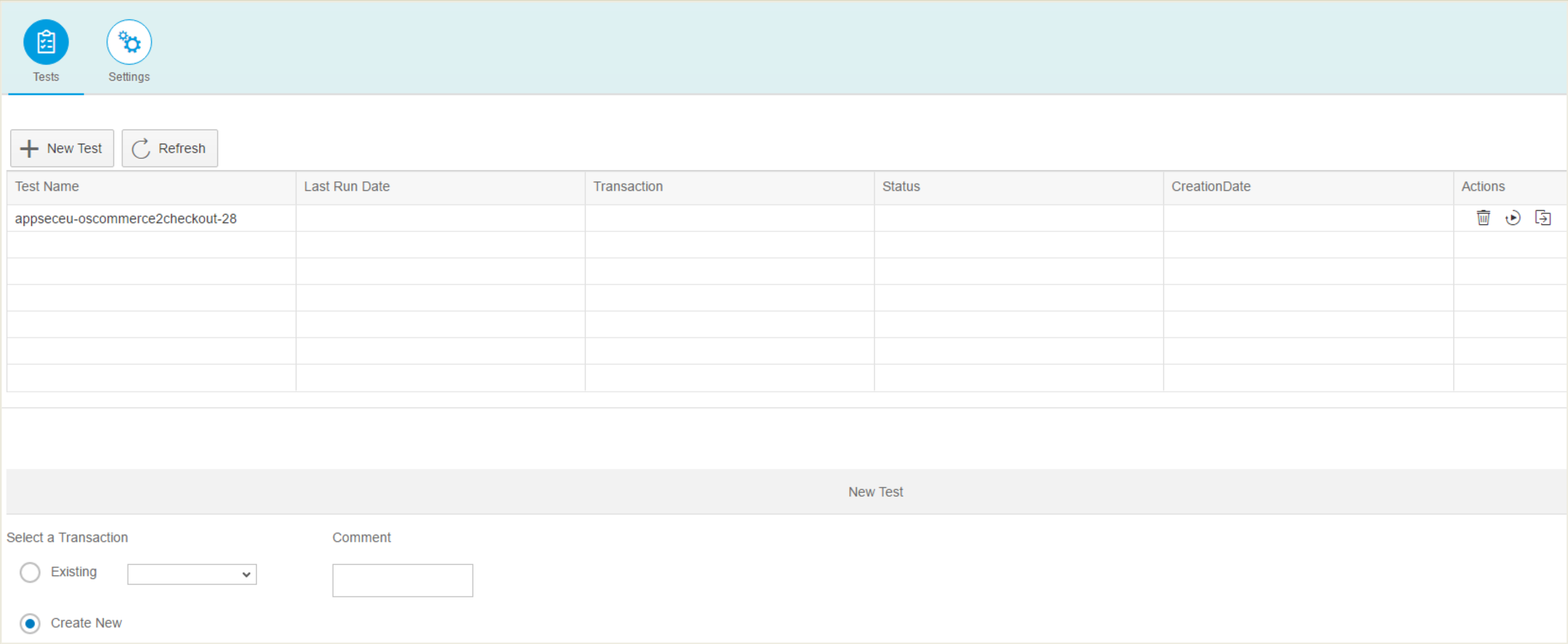
[NDSS 2016] A. Sudhodanan, A. Armando, R. Carbone, L. Compagna, Attack Patterns for Black-Box Security Testing of Multi-Party Web Applications

Demo

Scenario: Cashier-as-a-Service (CaaS)
SP_T: OsCommerce v2.3.4
TTP: 2checkout (sandbox)

Name	Attack Strategy	Precondition	Postcondition
RA3	REPLAY x FROM Session(U_M, SP_T) IN Session(U_M, SP_T)	TTP-SP $\in$ x.flow AND SU $\in$ x.labels	*Flag*(U_V, SP_T) e.g. “Your Order Has Been Processed!”

Demo UI: Create a New Test



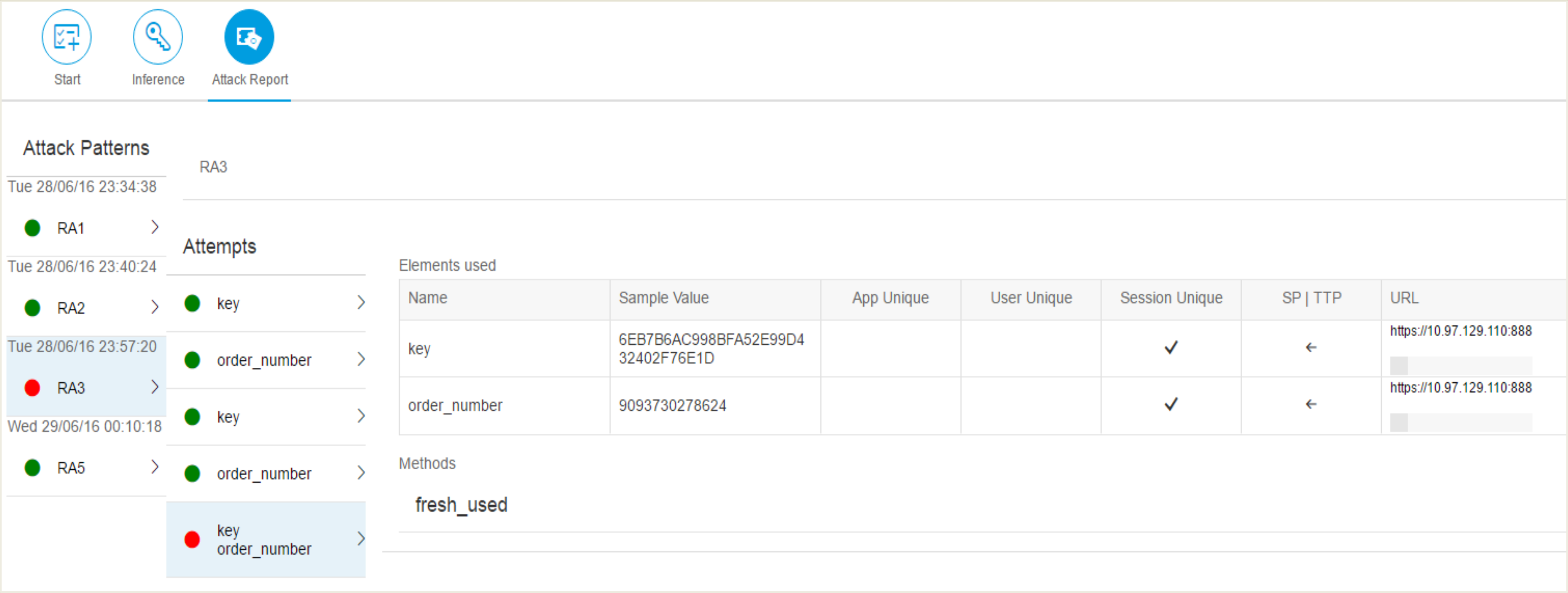
Demo UI: Inference Outcome

Start | Inference | Attack Report

Details (on)

Name	Sample Value	App Unique	User Unique	Session Unique	SP | TTP	URL	Location
email_address	user2%40blast.com		✓		↔	https://10.97.129.110:888	request.body
sid	901321725	✓			↔	https://sandbox.2checkou	request.body
first_name	User2		✓		↔	https://sandbox.2checkou	request.body
last_name	Two		✓		↔	https://sandbox.2checkou	request.body
street_address	2+Boulevard+de+Strasbc		✓		←	https://sandbox.2checkou	request.body
city	Strasbourg		✓		↔	https://sandbox.2checkou	request.body
zip	67000		✓		↔	https://sandbox.2checkou	request.body
email	user2%40blast.com		✓		↔	https://sandbox.2checkou	request.body

Demo UI: Attack Patterns Execution Outcome



Agenda

Introduction & Problem

- Multi-Party Web Applications (MPWAs)
- Logical Vulnerabilities in MPWAs
- Detecting Attacks caused by Logical Vulnerabilities

Observations & Solution

- Attacks to Attack Pattern
- Attack Pattern-based Security Testing

Results & Demo

Industrial Exploitation, Limitations & Future Work

Industrial Exploitation (preliminary)

Experimenting our prototype internally at SAP

Pilots with business units

- E.g., SAP Hybris e-commerce

Improving the usability of the prototype (e.g., UI): in-progress

Prototype availability

- Currently prototype available at SAP only
- However if you have a scenario you would like to validate reach out to us
- Delivery model still under discussion

Limitations and future directions

Coverage

- general issue for black-box techniques
- attack patterns can state precisely what they are testing
- still our approach is not complete
- can we reach practical full-coverage for replay attacks?

Observability

- our approach can observe client side communication
- server-to-server (S2S) communication is not considered
- what would we gain by adding S2S observability?

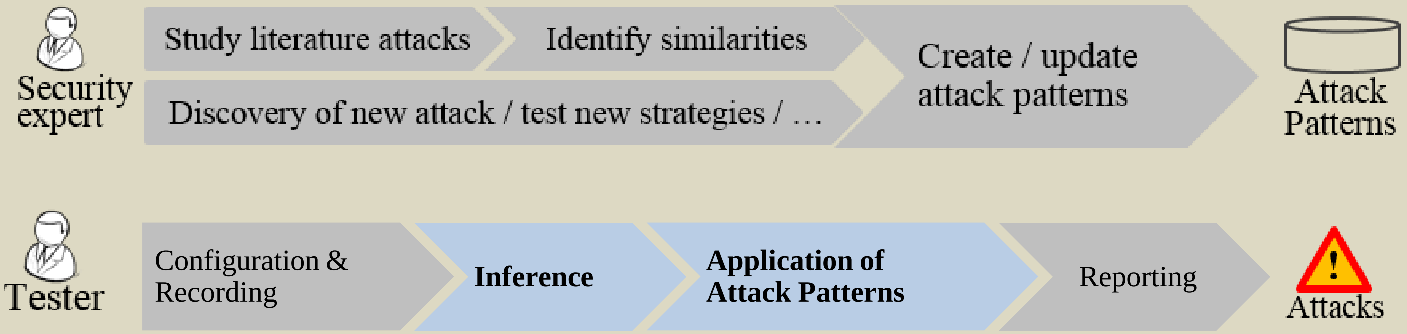
Conclusions

- Identified **7 attack patterns**
- Introduced a **black-box security testing framework** leveraging our attack patterns to discover vulnerabilities in the implementations of MPWAs
- Implementation based on **OWASP ZAP** (a widely-used open source penetration testing tool)
- Using our tool we discovered **21 previously-unknown vulnerabilities** in SSO, CaaS and **beyond**
- Industrial **exploitation** on-going

References

[1] Zhou, Y. and Evans, D. SSOScan: automated testing of web applications for single sign-on vulnerabilities. USENIX 2014
[2] Armando, A., Carbone, R., Compagna, L., Cuellar, J., and Tobarra, L. Formal Analysis of SAML 2.0 Web Browser Single Sign-On: Breaking the SAML-based Single Sign-On for Google Apps. FMSE 2008
[3] Bai, G., Lei, J., Meng, G., Venkatraman, S. S., Saxena, P., Sun, J., Liu, Y., and Dong, J. S. Authscan: Automatic extraction of web authentication protocols from implementations. NDSS 2013
[4] Pellegrino, G., and Balzarotti, D. Toward black-box detection of logic flaws in web applications. NDSS 2014
[5] Wang, R., Zhou, Y., Chen, S., Qadeer, S., Evans, D., and Gurevich, Y. Explicating SDKs: Uncovering assumptions underlying secure authentication and authorization. USENIX 2013
[6] Wang, R., Chen, S., and Wang, X. Signing me onto your accounts through facebook and google: A traffic-guided security study of commercially deployed single-sign-on web services. S&P 2012
[7] Sun, F., Xu, L., and Su, Z. Detecting logic vulnerabilities in e-commerce applications. NDSS 2014
[8] Bansal, C. and Bhargavan, K. and Maffeis, S. Discovering Concrete Attacks on Website Authorization by Formal Analysis. CSF, 2012

Thank You



Security expert
Study literature attacks
Identify similarities
Discovery of new attack / test new strategies / …
Create / update attack patterns
Attack Patterns
Tester
Configuration & Recording
Inference
Application of Attack Patterns
Reporting
Attacks