Attack Patterns for Black-Box Security Testing of Multi-Party Web Applications

Avinash Sudhodanan (sudhodanan@fbk.eu)
Alessandro Armando (armando@fbk.eu)
Roberto Carbone (carbone@fbk.eu)
Luca Compagna (luca.compagna@sap.com)

NDSS, San Diego, 22/02/2016

Multi-Party Web Applications (MPWAs)

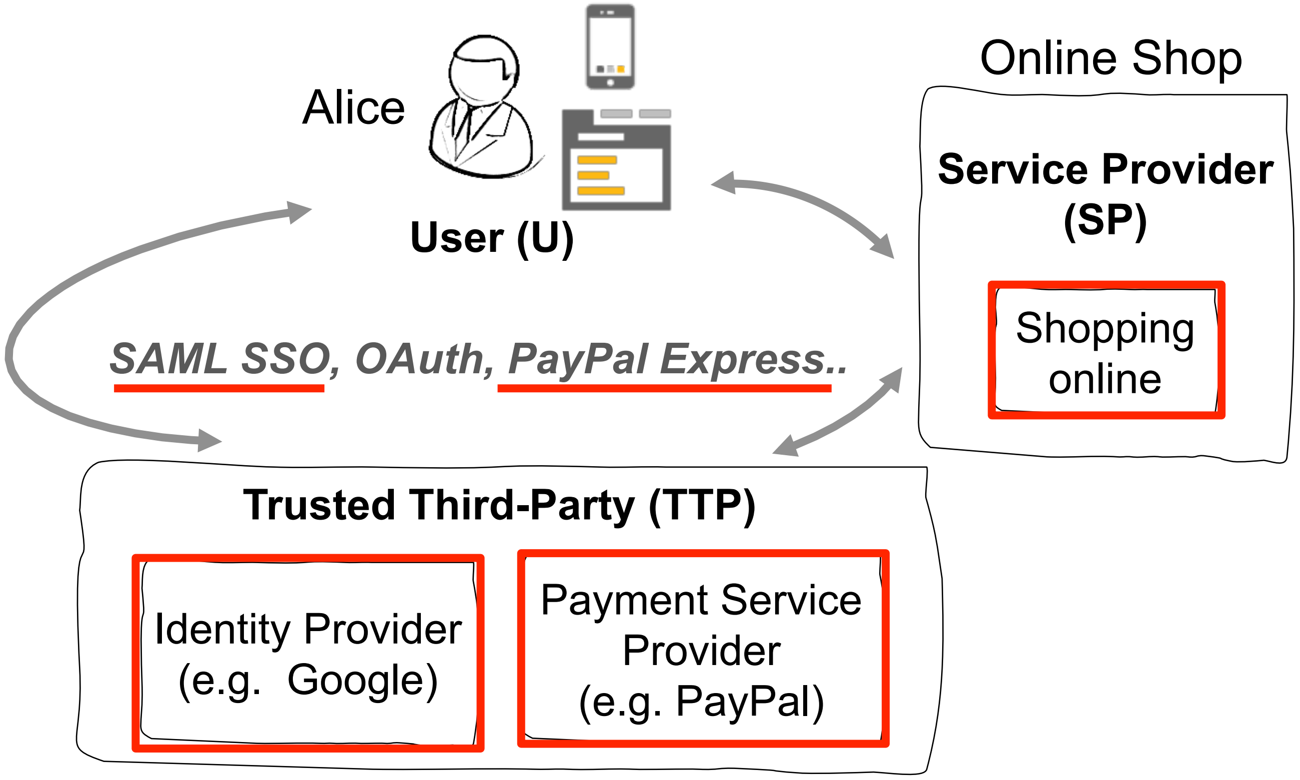
A **Service Provider** web app. relying on **Trusted Third-Parties** to deliver its services to **Users** through **web-based security protocols**

Examples

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

Popularity/Relevance

- 27% of top 1000 US websites supports Facebook SSO [1]
- 179+ million PayPal users worldwide

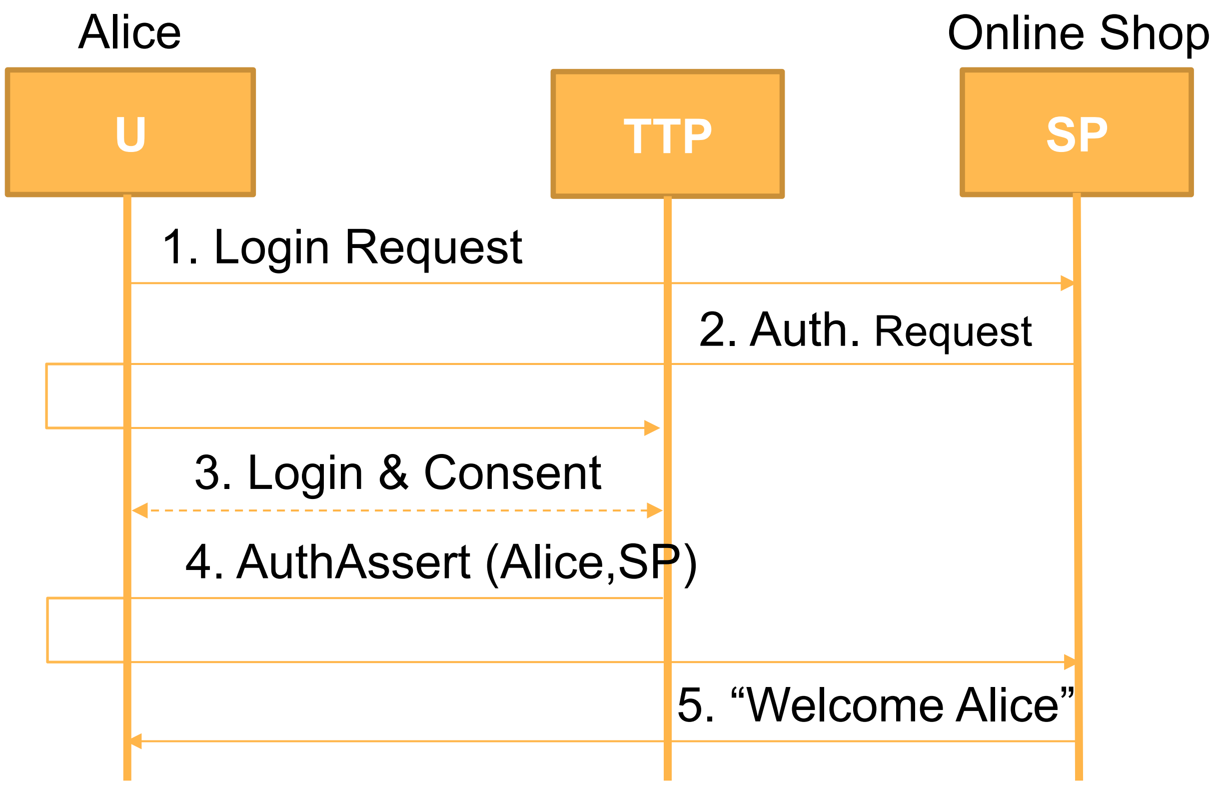


Multi-Party Web Applications (MPWAs)

A **Service Provider** web app. relying on **Trusted Third-Parties** to deliver its services to **Users** through **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 [1]
- 179+ million PayPal users worldwide

The **implementation** of the protocols underlying MPWAs is notoriously error-prone

Several Vulnerabilities Reported

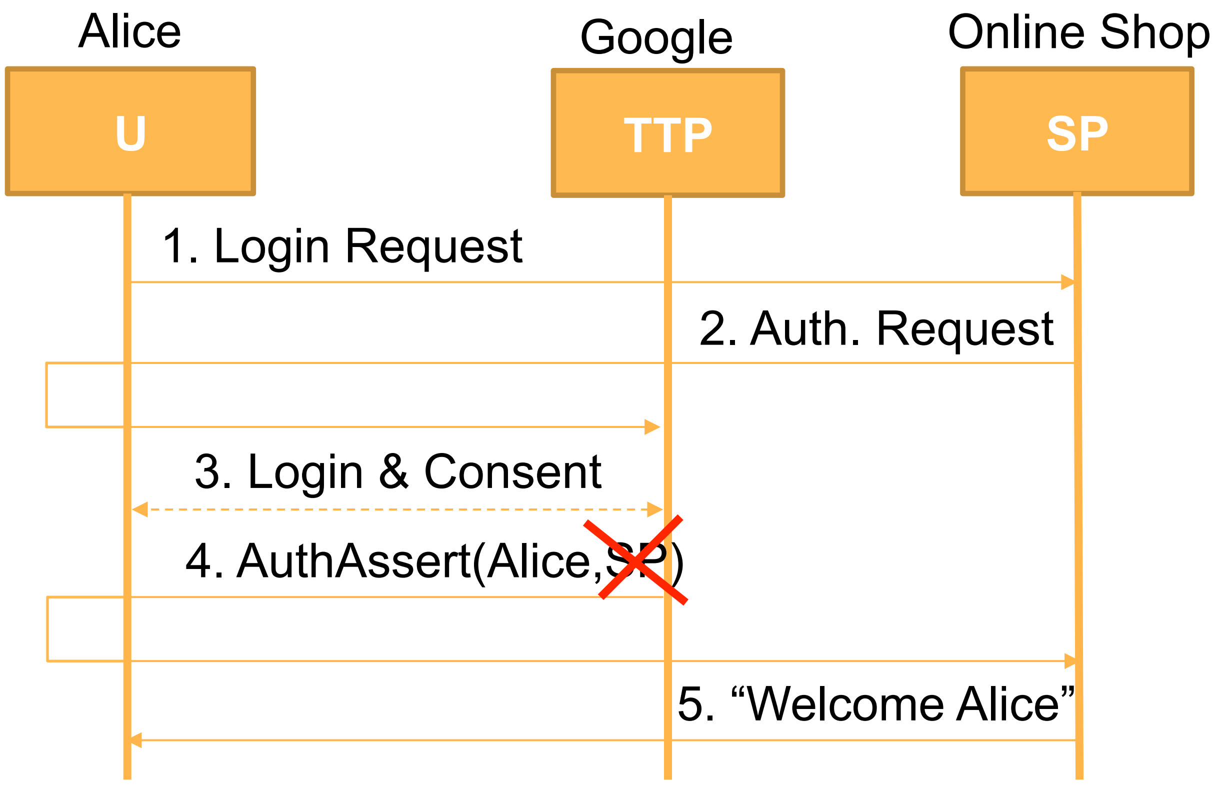
Many **vulnerabilities** discovered through a **variety of techniques** applied to **specific scenarios**

Tech. [Ref.]	Vulnerable MPWA	Attack	Attacker's Goal
FV [2]	SPs implementing Google's SAML SSO	Replay U_V's *AuthAssert* for SP_M in SP_T	Authenticate as U_V at SP_T
GB+FV [3]	developer.mozilla.com (SP) implementing BrowserID	Make U_V browser send request to SP_T with U_M's *AuthAssert*	Authenticate as U_M at SP_T
BB [4]	PayPal Express Checkout in OpenCart 1.5.3.1	Replay *Token* of transaction T_1 at SP_T during transaction T_2 at SP_T	Complete T_2 at SP_T
FV [5]	SPs implementing Facebook SSO	Replay U_V's *AccessToken* for SP_M in SP_T	Authenticate as U_V at SP_T
BB [6]	PayPal Payments Standard in osCommerce v2.3.1	Replay *PayeeId* of SP_M during transaction T at SP_T	Complete T at SP_T
WB [7]	Authorize.net credit card sim in baby products store	Replay *OrderId* of transaction T_1 at SP_T during transaction T_2 at SP_T	Complete T_2 at SP_T
FV [8]	CitySearch.com (SP) using Facebook SSO	Make U_V browser send request to SP_T with U_M's *AuthCode*	Authenticate as U_M at SP_T

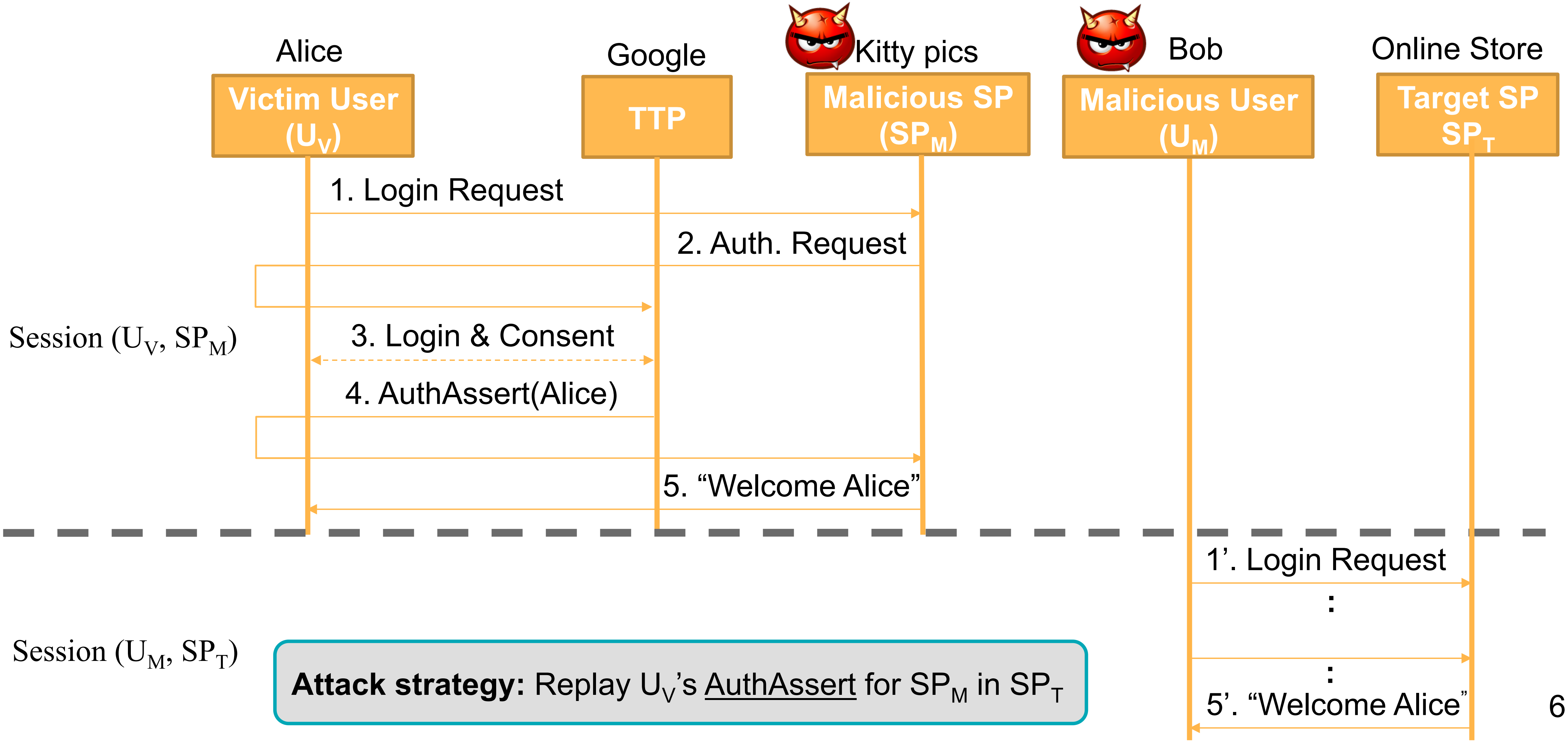
Legend- **FV**: Formal Verification, **GB**: Grey-Box Analysis, **BB**: Black-Box Analysis, **WB**: White-Box Analysis

SAML SSO: Example of vulnerable implementation

A man-in-the-middle attack against the SAML based SSO for Google Apps reported in [2]



SAML SSO: Example of vulnerable implementation



Attack strategy: Replay U_V's AuthAssert for SP_M in SP_T

Our Observation- I: attack strategies

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

Tech. [Ref.]	Vulnerable MPWA	Attack Strategy	Attacker's Goal
FV [2]	SPs implementing Google's SAML SSO	Replay U_V's *AuthAssert* for SP_M in SP_T	Authenticate as U_V at SP_T
GB+FV [3]	developer.mozilla.com (SP) implementing BrowserID	Make U_V browser send request to SP_T with U_M's *AuthAssert*	Authenticate as U_M at SP_T
BB [4]	PayPal Express Checkout in OpenCart 1.5.3.1	Replay *Token* of transaction T_1 at SP_T during transaction T_2 at SP_T	Complete T_2 at SP_T
FV [5]	SPs implementing Facebook SSO	Replay U_V's *AccessToken* for SP_M in SP_T	Authenticate as U_V at SP_T
BB [4]	PayPal Payments Standard in osCommerce v2.3.1	Replay *PayeeId* of SP_M during transaction T at SP_T	Complete T at SP_T
WB [6]	Authorize.net credit card sim in baby products store	Replay *OrderId* of transaction T_1 at SP_T during transaction T_2 at SP_T	Complete T_2 at SP_T
FV [7]	CitySearch.com (SP) using Facebook SSO	Make U_V browser send request to SP_T with U_M's *AuthCode*	Authenticate 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: preconditions

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

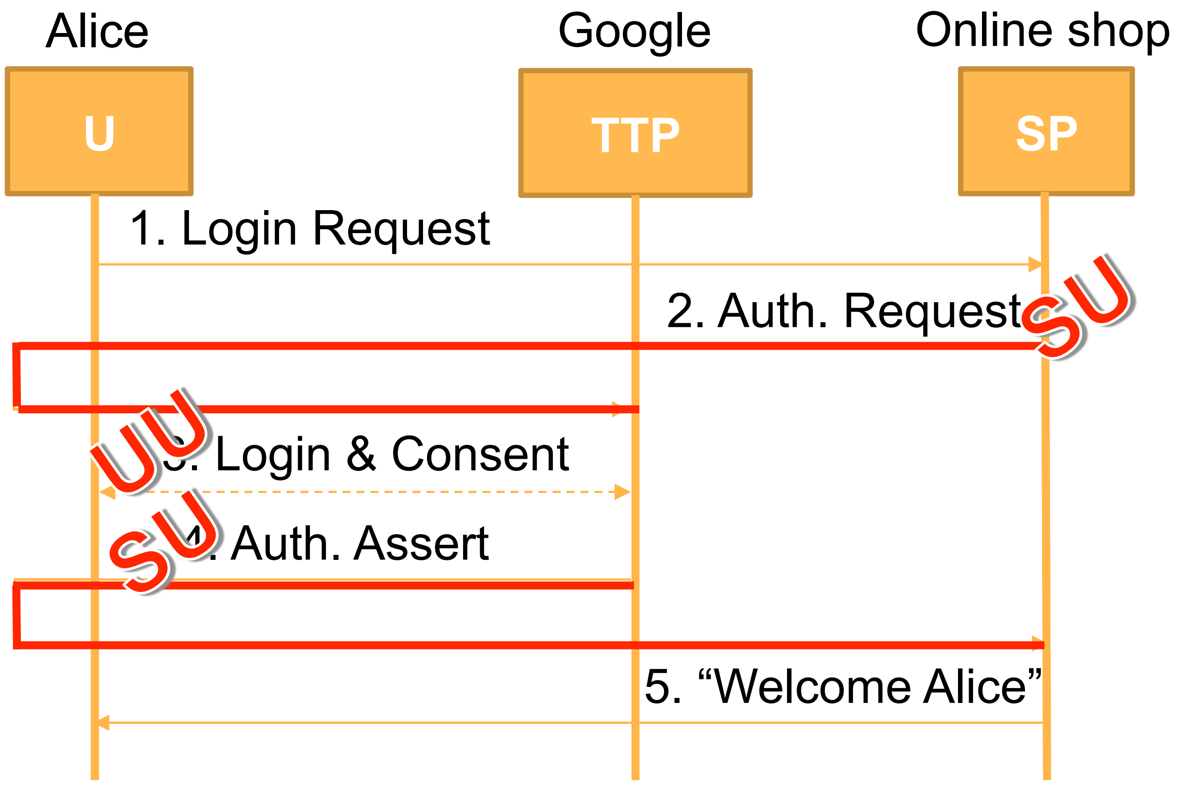
- **Syntactic/Semantic properties** of HTTP elements [8]

Property	Label
User Unique	UU
Session Unique	SU

:

- **Data flow 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



Can we understand from the HTTP traffic of the underlying protocol which attack strategy to be applied?

Our Observation-III: threat model

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

The thread model: Attacker can play the role of a **User** and/or a **Service Provider**

Is this threat model general enough for our purpose? Any added value by considering browser history attacker?

From Attacks to Attack Patterns

Tech. [Ref.]	Vulnerable MPWA	Attack Strategy	Attacker's Goal
FV [2]	SPs implementing Google's SAML SSO	Replay U_V's *AuthAssert* for SP_M in SP_T	Authenticate as U_V at SP_T
GB+FV [3]	developer.mozilla.com (SP) implementing BrowserID	Make U_V browser send request to SP_T with U_M's *AuthAssert*	Authenticate as U_M at SP_T
BB [4]	PayPal Express Checkout in OpenCart 1.5.3.1	Replay *Token* of transaction T_1 at SP_T during transaction T_2 at SP_T	Complete T_2 at SP_T
FV [5]	SPs implementing Facebook SSO	Replay U_V's *AccessToken* for SP_M in SP_T	Authenticate as U_V at SP_T
BB [4]	PayPal Payments Standard in osCommerce v2.3.1	Replay *PayeeId* of SP_M during transaction T at SP_T	Complete T at SP_T
WB [6]	Authorize.net credit card sim in baby products store	Replay *OrderId* of transaction T_1 at SP_T during transaction T_2 at SP_T	Complete T_2 at SP_T
FV [7]	CitySearch.com (SP) using Facebook SSO	Make U_V browser send request to SP_T with U_M's *AuthCode*	Authenticate as U_M at SP_T

From Attacks to Attack Patterns: one example

Tech. [Ref.]	Vulnerable MPWA	Attack Strategy	Attacker's Goal
FV [2]	SPs implementing Google's SAML SSO	Replay U_V's *AuthAssert* for SP_M in SP_T	Authenticate as U_V at SP_T
GB+FV [3]	developer.mozilla.com (SP) implementing BrowserID	Make U_V browser send request to SP_T with U_M's *AuthAssert*	Authenticate as U_M at SP_T
BB [4]	PayPal Express Checkout in OpenCart 1.5.3.1	Replay *Token* of transaction T_1 at SP_T during transaction T_2 at SP_T	Complete T_2 at SP_T
FV [5]	SPs implementing Facebook SSO	Replay U_V's *AccessToken* for SP_M in SP_T	Authenticate as U_V at SP_T
BB [4]	PayPal Payments Standard in osCommerce v2.3.1	Replay *PayeeId* of SP_M during transaction T at SP_T	Complete T at SP_T
WB [6]	Authorize.net credit card sim in baby products store	Replay *OrderId* of transaction T_1 at SP_T during transaction T_2 at SP_T	Complete T_2 at SP_T
FV [7]	CitySearch.com (SP) using Facebook SSO	Make U_V browser send request to SP_T with U_M's *AuthCode*	Authenticate as U_M at SP_T

Ref.	Vulnerable MPWA	Attack Strategy	Attacker's Goal
FV [2]	SPs implementing Google's SAML SSO	Replay U_V's *AuthAssert* for SP_M in SP_T	Authenticate as U_V at SP_T
FV [5]	SPs implementing Facebook SSO	Replay U_V's *AccessToken* for SP_M in SP_T	Authenticate as U_V at SP_T

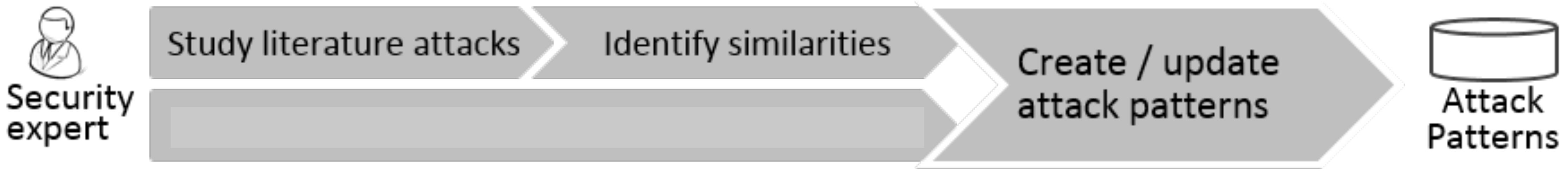
Id	Attack Strategy (Formalized)
1	REPLAY $AuthAssert$ FROM (U_V, SP_M) IN (U_M, SP_T)
2	REPLAY $AccessToken$ FROM (U_V, SP_M) IN (U_M, SP_T)

Name	Attack Strategy (Formalized)	Precondition	Postcondition
RA1	REPLAY x FROM (U_V, SP_M) IN (U_M, SP_T)	$(\text{TTP-SP} \in x.\text{flow AND } (\text{SU}\|\text{UU}) \in x.\text{labels})$	(U_V, SP_T) e.g. "Welcome Alice"

Attack Patterns

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

Approach



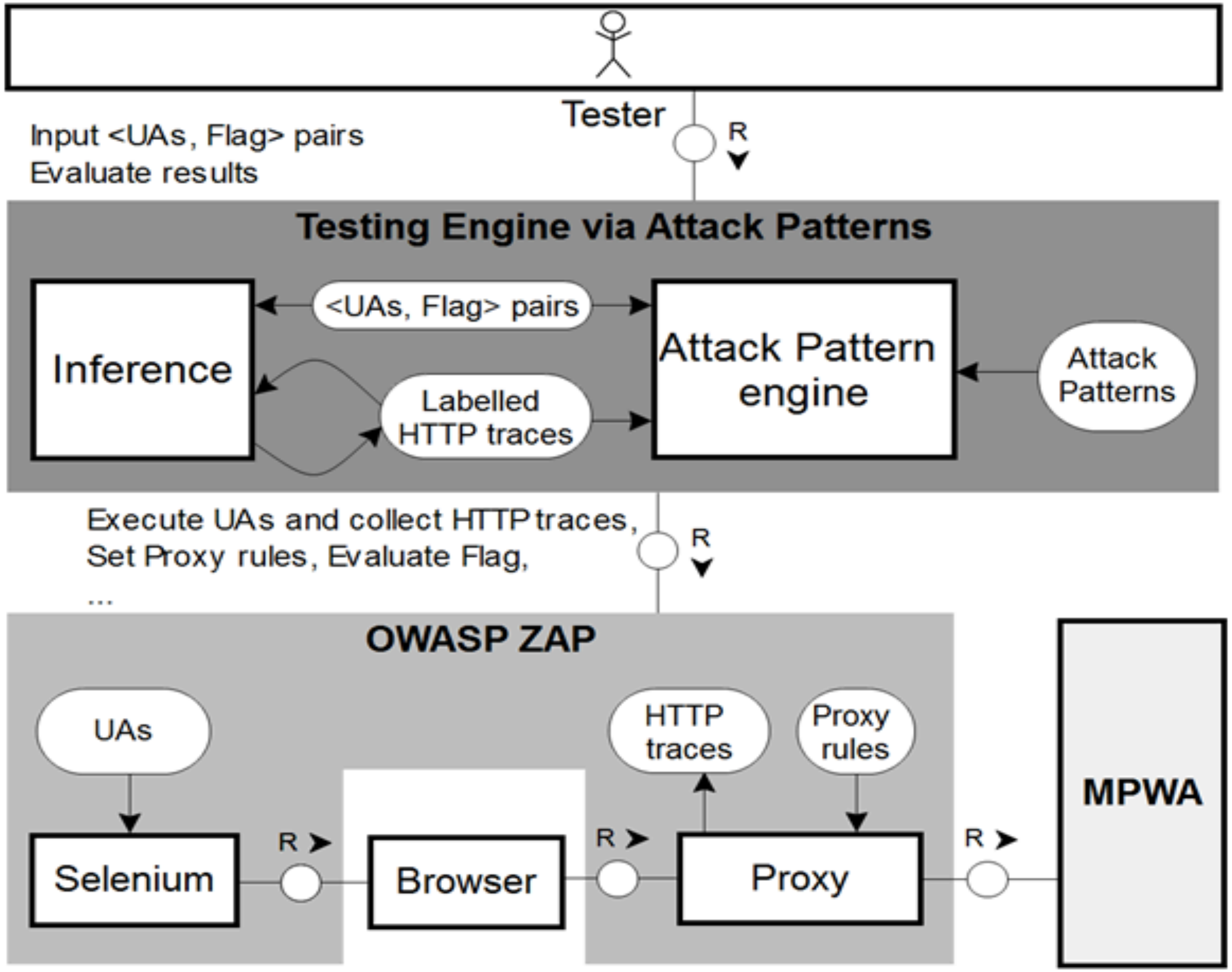
Knowledge of the security expert is **encapsulated** in **attack patterns**

- 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 **actions** e.g. **replay** an **element** from one protocol run in another
- Check **postconditions**

Implementation



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** found other scenarios e.g. **CaaS**	All SPs	Stripe Checkout	RA4 (*DataKey, Token*)	✓
	open.sap.com	Gmail (reg. via email)	LCSRF *(Act. Link)*	✓
Same attack in another protocol of same scenario	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)*	

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**

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?

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

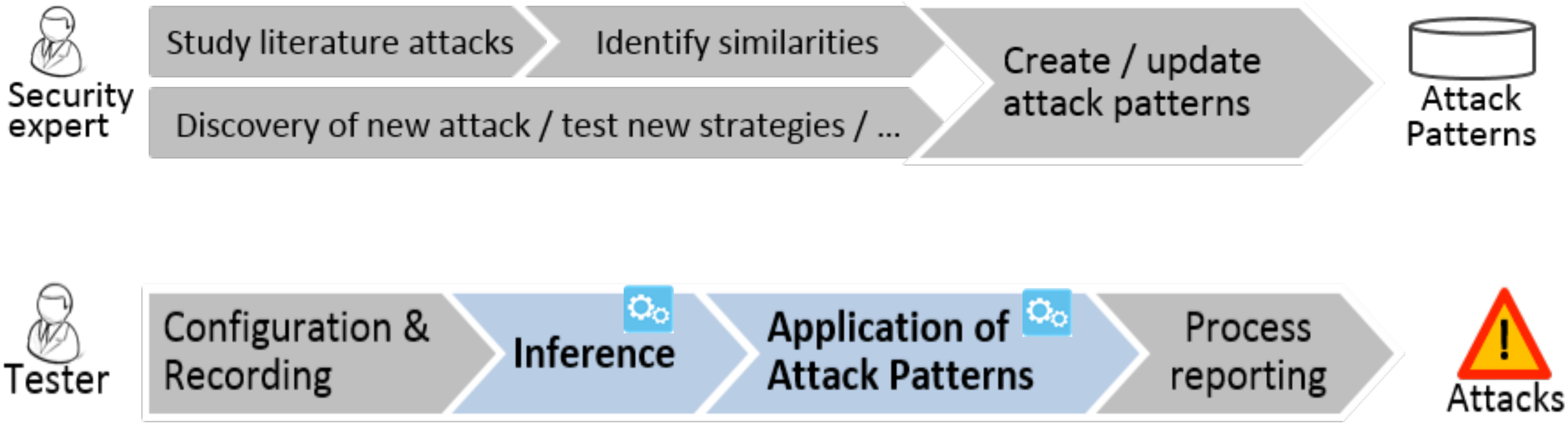
[6] Sun, F., Xu, L., and Su, Z. Detecting logic vulnerabilities in e-commerce applications. NDSS 2014

[7] Bansal, C. and Bhargavan, K. and Maffeis, S. Discovering Concrete Attacks on Website Authorization by Formal Analysis. CSF, 2012

[8] 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

Thank You

sudhodanan@fbk.eu



Backup slides

Example Attack Pattern: RA1

Name: RA1

Threat Model: Web Attacker

Inputs: $\mathrm{UAs}(U_V, SP_M)$, $\mathrm{LHT}(U_V, SP_M)$,
$\mathrm{UAs}(U_M, SP_T)$, $\mathrm{Flag}(U_V, SP_T)$

Preconditions: At least one element x in $\mathrm{LHT}(U_V, SP_M)$ is such that (TTP-SP $\in$ x.flow AND $(SU|UU) \in$ x.labels)

Actions:

For each x such that preconditions hold

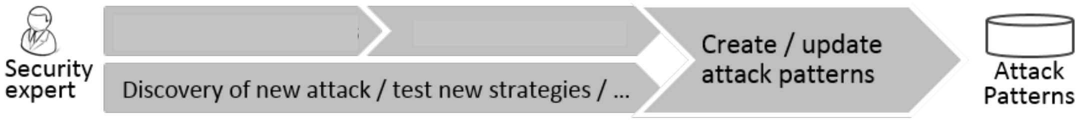
e = extract(x, $\mathrm{UAs}(U_V, SP_M)$)

HTTP_logs = replay(x, e, $\mathrm{UAs}(U_M, SP_T)$)

Check Postconditions;

Postconditions: Check $\mathrm{Flag}(U_V, SP_T)$ in HTTP_logs
Report(e, $\mathrm{UAs}(U_M, SP_T)$, $\mathrm{Flag}(U_V, SP_T)$)

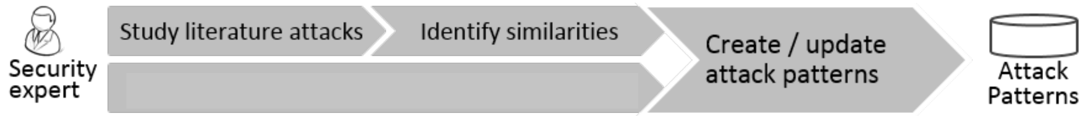
Custom Strategies



Threat Model: Browser History of victim user (U_V) is available to Attacker

Name	Attack Strategy	Precondition	Postcondition
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)

Complex Attack Patterns

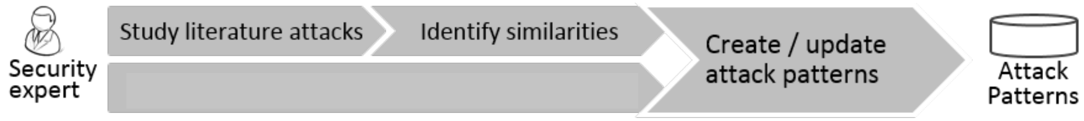


#	Vulnerable MPWA	Description of the Attack	Attacker's Goal
9	Github (TTP) implementing OAuth 2.0 Authorization Code flow-based SSO [1, Bug 2]	Replace the value of $RedirectURI$ to MALICIOUSURI in the session between U_V and SP_M to obtain $AuthCode$ of U_V and replay this $AuthCode$ in the session between U_M and SP_T	Authenticate as U_V at SP_T
10	*SP*s implementing Facebook SSO [2]	Replace the value of $RedirectURI$ to MALICIOUSURI in the session between U_V and SP_M to obtain $AccessToken$ of U_V and replay this $AccessToken$ in the session between U_M and SP_T	Authenticate as U_V at SP_T

Id	Attack Strategy
9	REPLAY $AuthCode$ FROM S IN (U_M, SP_T) where $S =$ REPLACE $RedirectURI$ WITH MALICIOUSURI IN (U_V, SP_T)
10	REPLAY $AccessToken$ FROM S IN (U_M, SP_T) where $S =$ REPLACE $RedirectURI$ WITH MALICIOUSURI IN (U_V, SP_T)

Name	Attack Strategy	Precondition	Postcondition
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)

LCSRF Attack Pattern

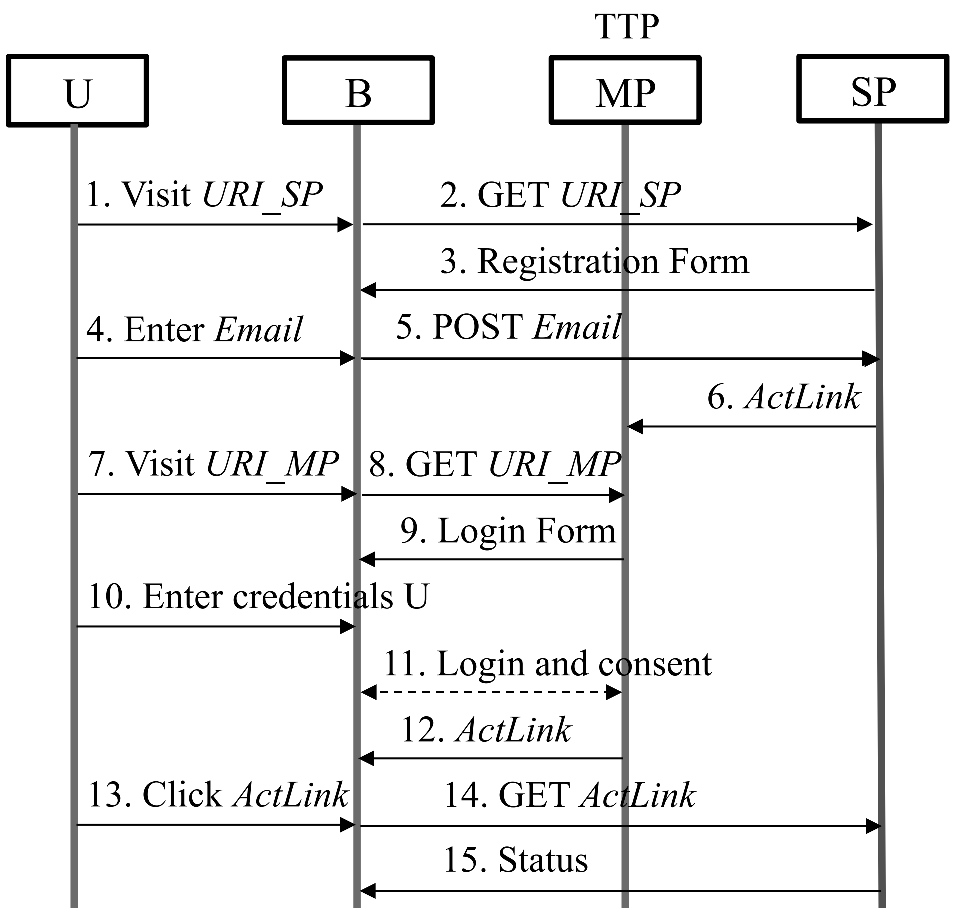


#	Vulnerable MPWA	Description of the Attack	Attacker's Goal
7	developer.mozilla.com (SP) implementing BrowserID [24, §6.2]	Make U_V browser send request to SP_T with U_M's $AuthAssert$	Authenticate as U_M at SP_T
8	CitySearch.com (SP) using Facebook SSO (OAuth 2.0 Auth. Code Flow) [25, §V.C]	Make U_V browser send request to SP_T with U_M's $AuthCode$	Authenticate as U_M at SP_T

Id	Attack Strategy
7	REPLACE x WITH REQUEST-OF $AuthAssert$ FROM (U_M, SP_T) IN $[U_M$ SEND $x]$
8	REPLACE x WITH REQUEST-OF $AuthCode$ FROM (U_M, SP_T) IN $[U_M$ SEND $x]$

Name	Attack Strategy	Precondition	Postcondition
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)

Beyond SSO and CaaS scenario: Reg. via email



Our Observation-III: threat model

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

Configuration		
One TTP	TTP	The TTP which is considered non-malicious
Two SPs	SP_T	The target SP who has a protocol integration with TTP
	SP_M	Another SP that has the same protocol implementation as SP_T
Two Us	U_V	The user representing a potential victim
	U_M	The user representing a malicious attacker

The thread model: Attacker can play the role of a **User** and/or a **Service Provider**

This threat model is general enough to detect the type of attacks we considered !

Our Observation-III: threat model

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

The thread model: Attacker can play the role of a **User** and/or a **Service Provider**

Is this threat model general enough for our purpose?